



ADVANCED DATABASES MODULE MANUAL 2024 (First Edition: 2024)

This manual enjoys copyright under the Berne Convention. In terms of the Copyright Act, no 98 of 1978, no part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any other information storage and retrieval system without permission in writing from the proprietor.



The Independent Institute of Education (Pty) Ltd is registered with the Department of Higher Education and Training as a private higher education institution under the Higher Education Act, 1997 (reg. no. 2007/HE07/002). Company registration number: 1987/004754/07.

DID YOU KNOW?

Student Portal

The full-service Student Portal provides you with access to your academic administrative information, including:

- an online calendar,
- timetable,
- academic results,
- module content,
- financial account, and so much more!

Module Guides or Module Manuals

When you log into the Student Portal, the 'Module Information' page displays the 'Module Purpose' and 'Textbook Information' including the online 'Module Guides or 'Module Manuals' and assignments for each module for which you are registered.

Supplementary Materials

For certain modules, electronic supplementary material is available to you via the 'Supplementary Module Material' link.

Module Discussion Forum

The 'Module Discussion Forum' may be used by your lecturer to discuss any topics with you related to any supplementary materials and activities such as ICE, etc.

To view, print and annotate these related PDF documents, download Adobe Reader at following link below:

www.adobe.com/products/reader.html

IIE Library Online Databases

The following Library Online Databases are available. These links will prompt you for a username and password. Use the same username and password as for student portal. Please contact your librarian if you are unable to access any of these. Here are links to some of the databases:

Library Website	This library website gives access to various online resources and study support guides [Link]
LibraryConnect (OPAC)	The Online Public Access Catalogue. Here you will be able to search for books that are available in all the IIE campus libraries. [Link]
EBSCOhost	This database contains full text online articles. [Link]
EBSCO eBook Collection	This database contains full text online eBooks. [Link]
SABINET	This database will provide you with books available in other libraries across South Africa. [Link]
DOAJ	DOAJ is an online directory that indexes and provides access to high quality, open access, peer-reviewed journals. [Link]
DOAB	Directory of open access books. [Link]
IIESPACE	The IIE open access research repository [Link]
Emerald	Emerald Insight [Link]
HeinOnline	Law database [Link]
JutaStat	Law database [Link]

Table of Contents

Using this Manual.....	6
Introduction	7
Module Resources	8
Module Purpose	9
Module Outcomes	9
Glossary of Key Terms for this Module	10
Learning Unit 1: Fundamentals of Databases	12
1 Overview of Basic Database Concepts.....	12
2 Databases.....	12
3 Revision Exercises.....	44
4 Practical Activities	44
Learning Unit 2: Administering a Database.....	46
1 Introduction	46
2 Create a Database	46
3 Practical Exercises.....	63
Learning Unit 3: Creating Other Database Objects	64
1 Creating Other Database Objects.....	64
2 Create Indexes.....	64
3 Practical Exercises.....	74
Learning Unit 4: Getting Started with PL/SQL programming	75
1 Introduction	75
2 Use PL/SQL Blocks	75
3 Practical Exercises.....	101
Learning Unit 5: Using Variables and Lexical Units	103
1 Introduction	103
2 Use PL/SQL Variables and Data Types.....	103
3 Practical Exercises.....	114
Learning Unit 6: Using Control Structures	115
1 Introduction	115
2 Use PL/SQL Operators and Expressions	116
1. Introduction	Error! Bookmark not defined.
3 Practical Exercises.....	130
Learning Unit 7: Handling PL/SQL Exceptions	132
1 Introduction	132
2 Handle System-Defined Exceptions	132
3 Practical Exercises.....	138
Learning Unit 8: Creating PL/SQL Subprograms	139
1 Introduction	139
2 Use Stored Subprograms	140
3 Practical Exercises.....	157
Learning Unit 9: Advanced Interface Methods and Security	158
1 Introduction	158
2 Execute Procedures Overview.....	158
3 Revision Exercises.....	175
4 Recommended Additional Reading	175

Bibliography	176
Intellectual Property	177

Using this Manual

This manual has been developed to meet the specific objectives of the module and uses a number of different sources. It functions as a stand-alone resource for this module and no prescribed textbook or material is therefore required. There may, however, be occasions when additional readings are also recommended to supplement the information provided. Where these are specified, please ensure that you engage with the reading as indicated.

To succeed in this module, you will need to:

- Attend lectures/online sessions;
- Engage with the material and activities on Learn, together with the prescribed material contained in this manual, which may include additional readings.

Introduction

Welcome to the Advanced Databases module. This module covers database implementation and management using Oracle Express™. You will gain enormously from this module, as skills in database design are often neglected in programming courses. Ninety percent of most business applications have a database working in the background. If you never become involved in database administration, design, and/or database development, it can lead to a real loss in development in the IT world.

After you have completed this module, you should be proficient in going into any small to medium business (or a business unit of a large corporate), analysing the data needs of the business, and, thereafter, designing, building, and implementing a database using any Database Management System (DBMS) software specified. The skills learned in Oracle XE™ are easily transferable to Access, SQL Server, etc. The emphasis of the module is to develop strong skills in Oracle. Oracle skills are one of the most sought-after skills for programmers.

This is a practical module and is best learnt by “doing”. Questions and exercises are provided at the end of each learning unit. If you need more exercises, please feel free to refer to the module manual prescribed in the bibliography section. These texts also have many examples.

The module has nine learning units. You will need to gain adequate experience in the field to supplement the examples given in the module manual. Your lecturer will provide different approaches and examples for the different concepts. This guide is designed as a definitive guide that traverses the breadth and depth of Oracle's flagship relational database management system.

Key Features:

Foundational Concepts: Delve into the fundamental principles of relational databases, SQL, and the architecture that underlies Oracle Database. Lay a solid groundwork to understand how data is organized, stored, and retrieved.

SQL Mastery: Unravel the power of Structured Query Language (SQL) in Oracle Database. From basic queries to advanced data manipulation and optimization techniques, this textbook provides a hands-on approach to mastering SQL.

PL/SQL Development: Explore the intricacies of Oracle's Procedural Language/Structured Query Language (PL/SQL). Learn how to create stored procedures, triggers, and functions to enhance the functionality and performance of your Oracle databases.

Database Design and Optimization: Gain insights into the art of designing efficient and normalized databases. Understand the principles of indexing, partitioning, and performance tuning to ensure your Oracle databases operate at peak efficiency.

Advanced Features: Dive into advanced features and capabilities of Oracle Database, including data security, backup and recovery strategies, and the integration of Oracle databases into broader enterprise architectures.

Module Resources

Please note that this manual is the prescribed material for this module. Refer to the *Module Resources* section in the **Module Outline document** for more information regarding recommended reading and other resources relevant to this module.

Module Purpose

The purpose of this module is to advance and develop your applied skills in database design and implementation within a commercial database management system.

Module Outcomes

MO1	Demonstrate the ability to create, manipulate and control database objects using SQL queries and control structures.
MO2	Describe procedural structured query language (PL/SQL) and manipulate data using PL/SQL coding language.
MO3	Define and apply exception handling techniques.
MO4	Create and use PL/SQL subprograms to extend PL/SQL functionality.
MO5	Demonstrate the ability to interface databases with an organisation's platform.
MO6	Apply database security principles within a given organisation's database setup.

Glossary of Key Terms for this Module

Term	Definition
Backup and Recovery	The process of creating copies of the database to prevent data loss and restoring the database to a previous state in case of failure
Commit	The process of making permanent the changes made during a transaction. Once committed, changes are visible to other transactions
Constraint	Rules defined on a table to control the kind of data that can be stored in a table. Examples include PRIMARY KEY, UNIQUE, CHECK, and FOREIGN KEY constraints
Data Dictionary	A set of tables and views that contains metadata, describing the structure of the database, including tables, columns, indexes, and constraints
Foreign Key	A column or set of columns in a table that refers to the primary key of another table. It establishes a link between two tables, enforcing referential integrity
Index	A database object that provides a fast access path to rows in a table, enhancing the speed of data retrieval
PL/SQL (Procedural Language/Structured Query Language)	Oracle's extension of SQL that adds procedural programming features. It is used for creating stored procedures, functions, and triggers in the Oracle Database
Primary Key	A column or a set of columns that uniquely identifies each row in a table. It enforces the entity integrity of the database
Redo Log	A set of files that records all changes made to data in the database. It is crucial for recovery in the event of a system failure
Rollback	The process of undoing changes made during a transaction that has not been committed. It restores the database to its state before the transaction began
Oracle Database	A relational database management system (RDBMS) developed by Oracle Corporation. It is widely used for managing and organizing large amounts of data
Schema	A collection of database objects (tables, views, etc.) owned by a specific user in Oracle. It provides a way to organize and manage database objects
SQL (Structured Query Language)	A standard programming language used to manage and manipulate relational databases. Oracle Database uses SQL for querying and updating data
SQL*Plus	A command-line interface for Oracle Database that allows users to interact with the database by entering SQL commands
Transaction	A sequence of one or more SQL statements that are executed as a single unit. Transactions ensure the consistency and integrity of the database

Trigger	A set of instructions that are automatically executed ("triggered") in response to certain events on a particular table or view in the database
View	A virtual table based on the result of a SELECT query. It provides a way to present data stored in one or more tables in a specific way without storing the actual data

Learning Unit 1: Fundamentals of Databases	
Learning objectives addressed in this learning unit: LO1: Discuss the different database models. LO2: Describe tiered architecture to manage data more effectively. LO3: Discuss relational database management systems. LO4: Compare different types of RDBMS applications. LO5: Discuss data warehousing. LO6: Distinguish between SQL*Plus and SQL Developer. LO7: Explain how SQL*Plus can submit SQL statements and PL/SQL blocks for execution. LO8: Discuss how the Oracle SQL developer can improve productivity. LO9: Discuss the various data types available. LO10: Formulate and use different DDL and DML SQL statements. LO11: Construct and explain how and why to use: <ul style="list-style-type: none"> • Aliases; • Set Operators; • Subqueries; • Joins. LO12: Compare the various functions supported by the GROUP BY function. LO13: Discuss the purpose of DCL SQL statements: <ul style="list-style-type: none"> • COMMIT; • ROLLBACK; • SAVEPOINT. 	My notes

1 Overview of Basic Database Concepts

In this learning unit, you will review the basic database concepts and identify the various components of Oracle database. In this topic, you will summarize the fundamentals of databases.

You want to implement advanced database technologies and features to support huge volumes of data. But before that, you may want to refresh your memory and summarize some fundamental database concepts so that it will be helpful for you to understand the concepts of Oracle database.

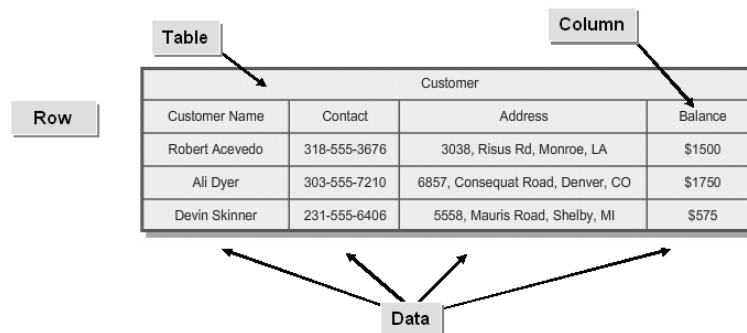
2 Databases

A database is an organized collection of data, usually in digital form. It's a collection of records and fields. A database is typically organized according to one of several general database models. The design and maintenance of a complex database needs

specialized skills and this task is performed by a database administrator (DBA). It's supported by tools provided either as part of the database management system (DBMS) or as a freestanding product.

A DBMS is an application that enables easy management of databases. By using a DBMS, users can store and retrieve structurally organized data.

Example



2.1 Manual Data Processing vs. Databases

Databases have definite advantages over a manual data processing system.

- **Data Redundancy:** Data stored in a manual data processing system does not have an organized structure; therefore, data repetition occurs. An effectively designed database can overcome data redundancy by using various data modeling techniques and tools.
- **Data Inconsistency:** A manual file processing system does not repopulate data modifications across the entire database, but databases enable you to update data consistently throughout the database.
- **File Incompatibility:** Files used in a file processing system may be of different formats, but databases overcome this by storing data in a specific structure. Databases support data sharing because they do not have file incompatibility issues.

2.2 Database Models

A database model is a theory or specification describing how a database is structured and used. There are several database models. In general, some of the common models are the flat-file model, which contains a single 2-dimensional array of data elements. Then we've got the hierarchical model where data is organized into a tree-like structure. We also have the network model, where data is organized using sets and records. The next one is the relational model, which is based on the first-order predicate logic in which the database is defined as a collection of predicates or functions or operators that return either true or false values. We also have the object relational model, which is similar to the relational model except that objects, classes

and inheritances are directly supported in database schemas. The last one is the star-schema model which is said to be the simplest style of data warehouse schema (R, 2015).

Here are some more details about the various database models.

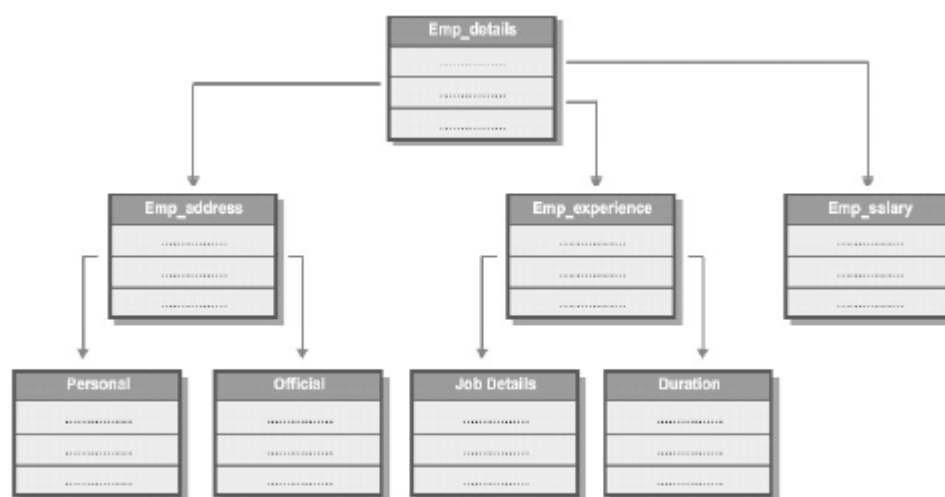
Flat file

This data model consists of a single, two dimensional array of data where all column members are assumed to have similar values and row members are related to each other.

ORD_NO	SLDATE	QTY	CUSTN	PART	REP
101	16-NOV-07	220	20503	40125	NO1
102	20-NOV-07	100	8802	40232	NO2
103	20-NOV-07	170	9989	40641	NO2

Flat File

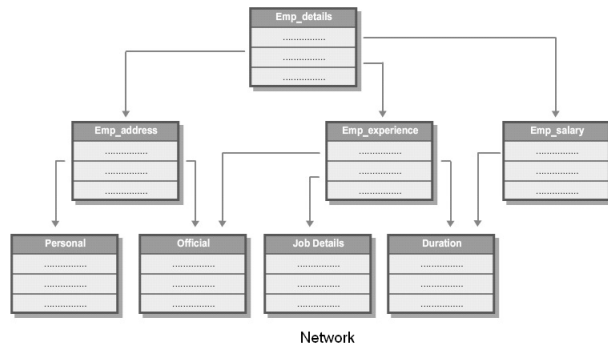
in this model are assimilated into various levels in which files are arranged in layers or tiers, and data is accessed through predefined relationships. This model enhances efficiency, provides greater control over a database, and simultaneously reduces redundancy in the database.



Hierarchical

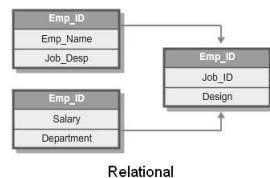
Network

This data model is based on the connectivity of data relationships, where multiple computers are used and information is stored and shared. It provides two alternative views of a database schema and subschema. A schema is a complete logical view of the entire database, and a subschema is a subordinate view of the database. Further, data relationships result in reduced redundancy.



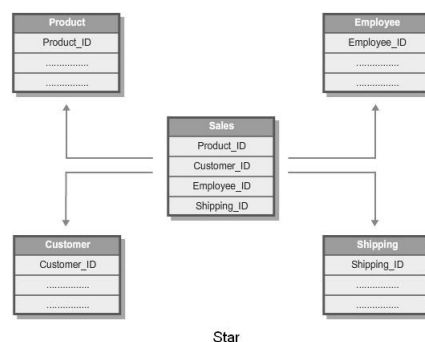
Relational

This data model stores logically related data consistently in the form of related tables. These tables are linked through common fields or columns. The data stored is independent of files and is managed by a central database engine that processes queries and manipulates data. Every row of data contains an identification key that identifies data uniquely and helps in reducing redundancy. Oracle database is one of the prominent relational database management systems used for managing data.



Star

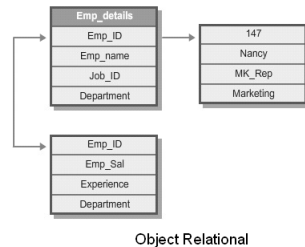
This data model consists of one or more fact tables with many dimension tables. The fact table consists of data recorded in events and the dimension table consists of attributes that describe the fact table data. This model reduces the number of tables and thus enhances the data retrieval process. The schema resembles a constellation of bright stars surrounded by dimmer stars, and so it's referred to as the star schema.



Object-relational

This model is designed to bridge the gap between the relational database model and the object-oriented model. It is integrated with the object-oriented programming language. The working approach is the same as that of the

relational databases provided the database acts as an object store for the software written using the object-oriented programming language. It supports complex data creation, type inheritance, and object behaviour that is related with program objects.



Object Relational

2.3 Types of Database Environment

At the broad level, you can classify a database environment into four types.

Type	Function
OLTP	Online Transaction Processing (OLTP) stores and manipulates data for handling queries and transactions. These transactions result in the creation of data or manipulation of existing data.
DSS	A Decision Support System (DSS), also known as a data warehouse, stores massive quantities of historical data for generating reports. You can retrieve, analyze, extract, and transform data based on your requirements
OLAP	Online Analytical Processing (OLAP) analyses data for data mining tasks such as budgeting or forecasting. For example, it supports services that analyses online information such as online phone numbers and email addresses.
Hybrid	A hybrid system acts as a multifunctional database, because it supports transactional processing, ad hoc querying, and batch processing.

2.4 Tiered Architecture

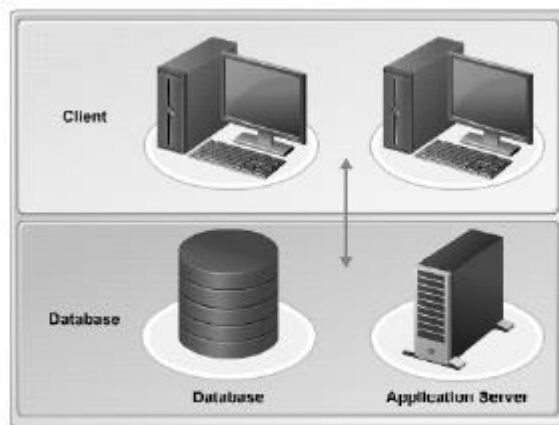
To manage data more effectively, a database is divided into layers or tiers known as the tiered architecture. In general, there are two types of tiered architecture. One is the two-tiered architecture and the other one is the multi-tiered architecture, also called the n-tiered architecture. The two-tiered architecture basically consists of a server and a client, where they both interact with each other. It is typically used in intranets. The multi-tiered architecture supports more than two layers or tiers, and the most widely used is the three-tier architecture consisting of three types of layers. One is the presentation layer, which is the topmost level of application. It displays information related to the user interface and retrieves data from different tiers. The next layer is the application layer, which is a logic tier. It coordinates application process commands, makes logical decisions and evaluations, and performs calculations (R, 2015). The last

among the layers is the data layer, with which information is stored and retrieved from databases.

Here are some more details about the database architecture types.

Two-tier Architecture

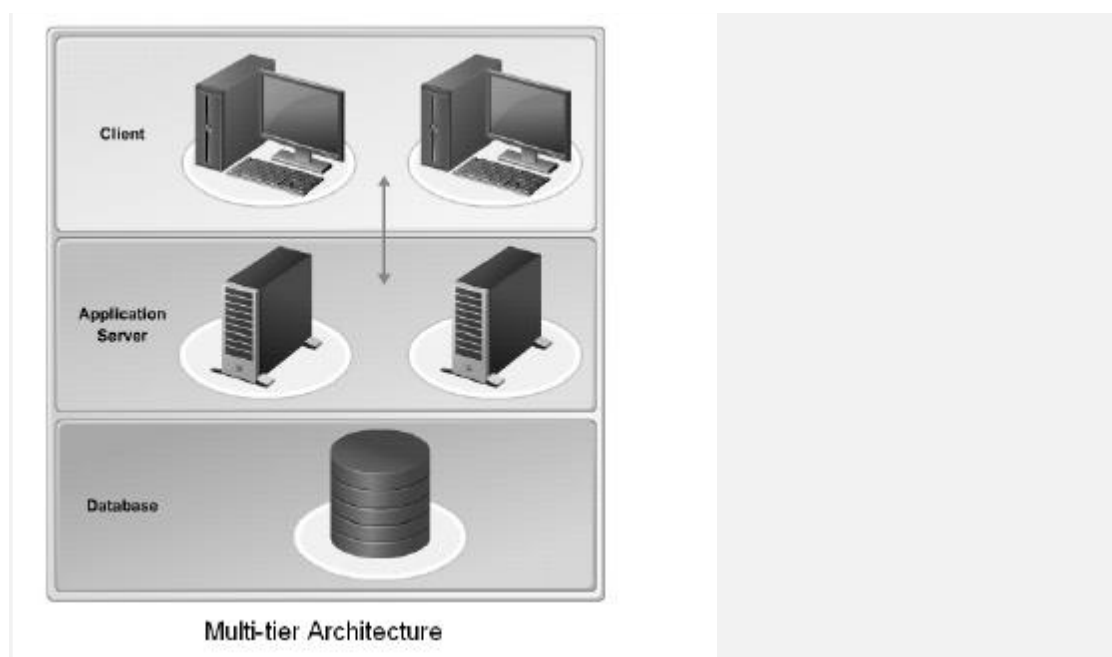
This is also known as the client-server architecture. In this type of architecture, a client sends a request and the server responds to it. One centralized server and multiple clients will be connected to the system through dedicated networks. Normally this is applicable in an intranet setup. Also, clients need a client software at their end. While a server is connected to more than one client at a time, each client is supplied with the necessary software to access the server. Therefore, any change in the server configuration may require a corresponding change in the individual client's configuration.



Two-tier Architecture

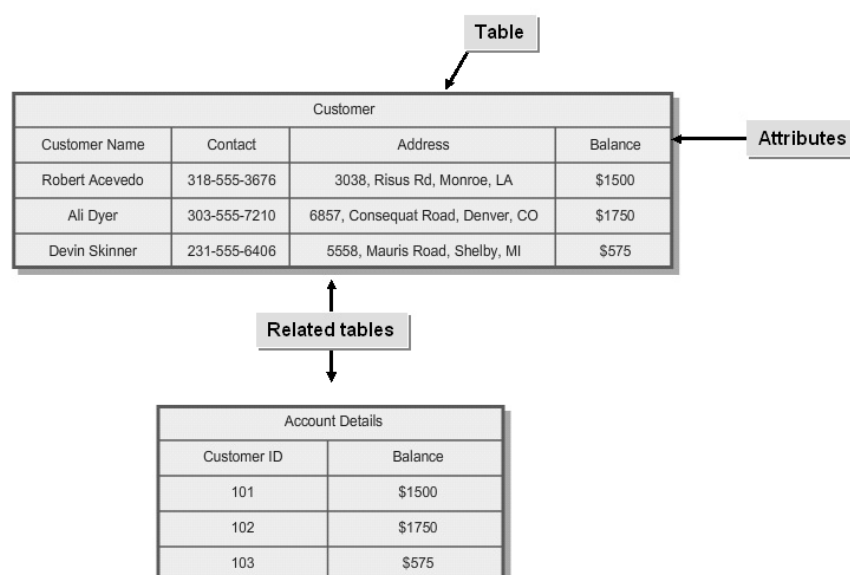
Multi-tier Architecture

This is also known as the N-tier architecture, where N is any number greater than two. It helps in implementing a multi-layered design where the database server is on the first tier, a centralized application is on the middle tier, and the database client application is on the third tier.



2.5 Relational Database Management Systems

A Relational Database Management System (RDBMS) is a database management system that uses a relational database model to hold data. Examples of RDBMS include Oracle, Sybase, SQL Server, DB2 and others. Oracle is an object-oriented database management system, where columns are represented either in a single value or in arrays. An RDBMS stores data in tables called relations. These relations are 2-dimensional representation of data where rows are called tuples and columns are called attributes. The data stored is independent and is managed by the central database engine that processes queries and manipulates data (R, 2015).



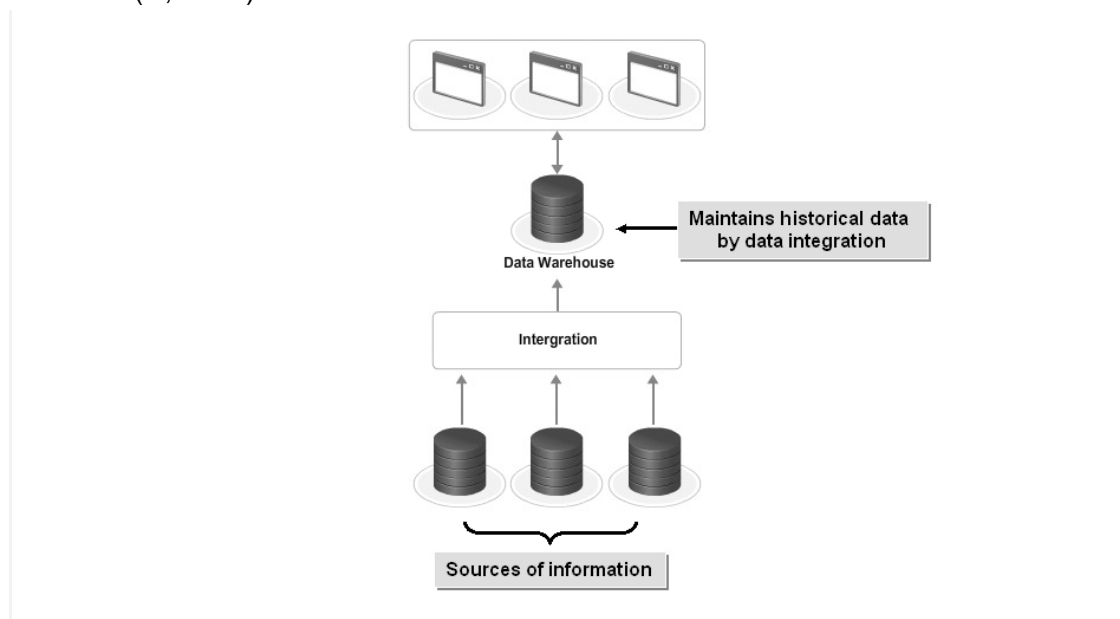
2.6 Data Warehouse

A data warehouse is a relational database management system designed for querying and analysis. It usually contains historical data derived from transactional data. It can also include data from other sources. A data warehouse environment can also include extract, transform, and load (ETL) capabilities. Data warehouses enable you to write simple queries, extract information within a minimum query response time, and implement OLAP.

Let's discuss the differences between an online transaction processing system (OLTP) and OLAP.

An OLTP has got a complex data structure, whereas an OLAP has got a multidimensional data structure.

An OLTP has only a few indexes, whereas an OLAP has multiple indexes. An OLTP is considered a normalized database, whereas an OLAP is considered a denormalized database(R, 2015).



2.7 RDBMS Applications

There are different types of RDBMS applications catering to various business and technical requirements.

RDBMS Application	Description
Oracle	Owned and developed by Oracle Corporation. Oracle database development tools include application express, forms, and SQL*Plus. This helps you to browse database objects, run SQL statements and scripts, and edit and debug PL/SQL statements.

SQL Server	Owned and developed by Microsoft Corporation. It helps you to manage data by enabling you to query, search, synchronize, report, or analyse data.
My SQL	Owned and developed by Sun Microsystems. It is an open source relational database management software. It can be modified.
Sybase	Owned and developed by Sybase. It provides the foundation for various enterprises, enabling them to manage and mobilize information from data centers.

2.8 Explore Oracle Database

You reviewed the fundamentals of a database. You now want to start using an Oracle database application. In this topic, you will explore the basic concepts of Oracle database, a relational database management system. Just as every application has its unique features and benefits that any new user will need to understand to use it effectively, it is very critical for you to understand the unique features and tools offered by Oracle database to manage data efficiently (R, 2015).

2.9 Advantages of Oracle Database

Oracle Database provides a wide set of advantages. It enables you to:

- Partition and compress tables to store more data.
- Protect and audit data to manage historical data.
- Integrate and manage the life cycle of enterprise information.
- Make use of the Oracle Maximum Availability architecture to achieve high availability at a lower cost and complexity.
- Eliminate time-consuming error-prone administrative tasks by using features such as self-management and automation, thereby enabling the DBA to focus on strategic business policies.
- Implement Oracle database on all operating systems such as Linux, UNIX, Microsoft Windows, Mainframes, and Mac. It provides seamless operational efficiency regardless of the operating system used.
- It also supports 64-bit architecture to add additional memory space for large applications.

2.10 Oracle Users

Oracle Database is used by various levels of users who can perform certain transactions, develop an application, or write efficient queries to recover data.

User Type	Description
End user	Retrieves data and uses it by initiating transactions or generating reports
Database developer	Designs and develops the interface. The application can be custom made for a specific purpose, such as web-based applications, or it can be designed to be sold in mass quantities as a standalone product.
Database Administrator (DBA)	Maintains a database management system. The DBA manages, monitors, tunes, and backs up data and recovers it in case of failure. The database management roles of the DBA include capacity planning, performance measurement, data recovery, data security, and user accounts management. In addition to this, the DBA also deals with hardware and software upgrades.
Database Operators	The responsibilities of Database Operators (DBO) are very similar to that of a DBA, except that they do not have deep-end knowledge and control of the internal workings of a database

2.11 The Oracle Program Categories

Oracle Database has various program categories that enable you to operate a database.

Program Category	Description
Application Development	Enables easy access to data in a database and allows an application to access many different data sources through ODBC (Open Database Connectivity). It further helps in database troubleshooting and error handling. In addition to this, it provides the Oracle SQL Developer tool, a Graphical User Interface (GUI), and SQL*Plus, a command line interface to interact with the database.
Configuration and Migration Tools	Provides ODBC and assists in configuring network connections.
Integrated Management Tools	Manages the security credentials on Oracle clients and servers.
Oracle Installation Products	Helps to install or uninstall Oracle configuration tools.
Warehouse Builder	Helps in ETL (Extract, Transform, Load), data quality, and the life cycle management of data

	and metadata, which is structured data that contains the context or characteristics about another data.
Database Control-<database name>	Helps to connect to the Enterprise Manager.

Oracle Interface

The Oracle interface is an application that enables you to connect to a Oracle database.

It helps you to create connections and link to various data sources such as Oracle, Access, MySQL, and SQL server (R, 2015).

Oracle SQL Developer

Oracle SQL Developer is a free graphical tool designed to improve your productivity and enables you to perform everyday database tasks efficiently.

SQL*Plus

SQL*Plus is an Oracle-developed application that provides developers and end users with a command-line interface to either connect or disconnect from the server. It is independent of any database versions or operating systems, allowing a single workstation to seamlessly connect to all available Oracle database servers.

The Oracle Enterprise Manager

The Oracle Enterprise manager application helps businesses to achieve the highest quality of service at the lowest IT operations cost. This approach blends broad application management and quality assurance solutions for Oracle technologies embedding Oracle packaged applications, quality Oracle Fusion Middleware, and Oracle database.

The Oracle Recovery Manager

The Oracle Recovery manager application is responsible for managing the backup and recovery process for all Oracle data formats. It works with the database server to detect and restore block-level corruption during backup and also keeps track of the various database procedures before and after backup (R, 2015).

2.12 Oracle Editions

Oracle Database includes various editions, each of which supports specific features.

Edition	Description
Enterprise	This Oracle edition is for servers with unlimited sockets. It provides industries with high performance, data warehousing and mining, scalability, and security. It can run on Windows, Linux, and Unix. In addition, it also protects a user from server and site failure, and reduces planned downtime.
Standard	This Oracle edition is for servers with up to four sockets. It comprises Oracle Real Application Clusters, which provide higher availability, performance, and security. This edition also runs well on Windows, Linux, and Unix. You can develop

	streamlined applications with Oracle Application Express, Oracle SQL Developer, and Oracle Data Access Components for Windows
Standard Edition One	This edition is for servers with up to two sockets. It provides security, availability, scalability, performance, and easy manageability. This runs comfortably on Windows, Linux, and Unix. You can develop streamlined applications with Oracle Application Express, Oracle SQL Developer, and Oracle Data Access Components for Windows.
Express	This edition is based on the Oracle Database 10g. Release 2 code and can be developed, deployed, and distributed free of cost. It is a starter database for developers, DBAs, independent software and hardware vendors, and educational institutions.

2.13 *Recap of Entity Relationship Diagrams*

An Entity-Relationship Diagram (ERD) is a visual representation of the data structure within a database or information system. It is a fundamental tool used in database design and modelling to depict the relationships between different entities or objects and their attributes. ERDs help in understanding how data is organized and how different elements in a system are related to each other.

There are three main components in an ERD:

Entities: Entities are the primary objects or concepts represented in a database. They can be real-world objects like customers, products, employees, or abstract concepts like invoices, orders, or transactions. In an ERD, entities are typically represented as rectangles or squares.

Attributes: Attributes are the properties or characteristics that describe entities. Each entity has a set of attributes that define it and provide details about it. For example, a "Customer" entity may have attributes like "CustomerID," "Name," "Email," and "Phone Number." Attributes are usually represented as ovals or ellipses connected to their respective entities by lines.

Relationships: Relationships represent how entities are connected or associated with each other. They illustrate the connections between entities and show how data is related in a database. Relationships are depicted as lines connecting entities, with optional labels or symbols to indicate the nature of the relationship. Common relationship types include:

One-to-One (1:1): This type of relationship indicates that one instance of an entity is associated with only one instance of another entity, and vice versa. For example, each person may have only one passport, and each passport belongs to only one person.

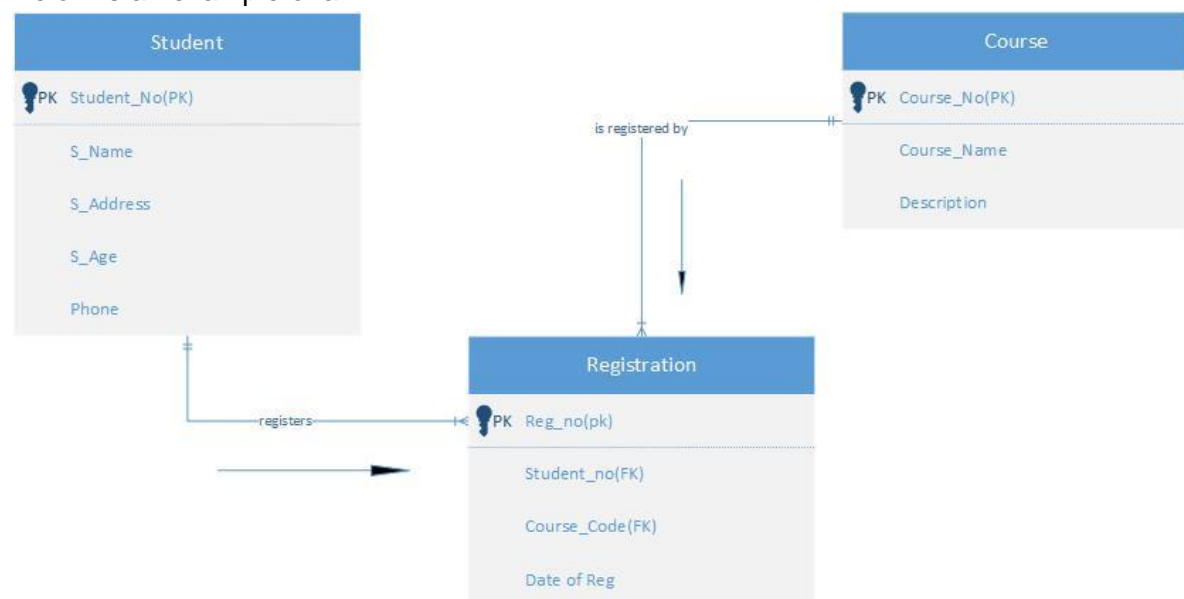
One-to-Many (1:N): In a one-to-many relationship, one instance of an entity is associated with multiple instances of another entity. For instance, one university department can have many students, but each student is associated with only one department.

Many-to-Many (N:N): This relationship type signifies that multiple instances of one entity are related to multiple instances of another entity. For example, in a library database, each book can be borrowed by multiple patrons, and each patron can borrow multiple books. This relationship usually requires the introduction of a junction or associative entity to represent it.

ERDs can also include additional elements like cardinality indicators (to specify how many instances are involved in a relationship) and participation constraints (to indicate whether participation in a relationship is optional or mandatory).

Overall, an ERD serves as a crucial tool for database designers, developers, and stakeholders to visualize and plan the structure of a database system, ensuring that data is organized efficiently and relationships are well-defined. It aids in the creation of a robust database schema that accurately represents the information requirements of an organization or system.

Below is an example of an ERD



2.14 Using SQL Commands to Work With Tables

So far in the course, you've identified the interface elements in an Oracle environment and the various components of the Oracle Database architecture. With this knowledge, you should feel ready to work comfortably with databases. One of the most common things you'll need to do is to create and access tables, which are information storage units in a database. In this section, you will use basic SQL commands to work with tables in a database.

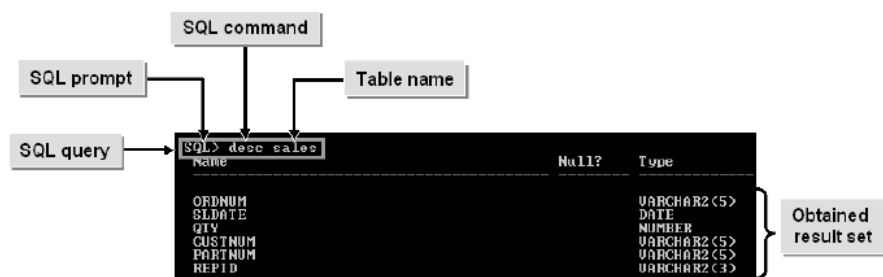
Can you talk to a native American in Mandarin Chinese? Probably not; similarly, you need to talk to your Oracle database using a language that the database can understand.that language is SQL, and any database administrator needs to be comfortable with it. So,let's use SQL commands to work with database tables (R, 2015).

2.15 Getting started with Oracle SQL Developer

Before we start creating objects in our database and issuing commands we need to understand how to access Oracle. There are 2 ways to do this.

1. **SQL Plus.:** SQL * Plus is an application from Oracle that provides a command line interface for DBAs, developers, and end users to connect and disconnect from the Oracle server. It is independent of any database version or operating system, allowing a single workstation to seamlessly connect to all available Oracle database servers. SQL * Plus runs both SQL and PL/SQL commands interactively. SQL commands are stored in special files called scripts, which can be saved, edited, and run from the prompt. Below is what it looks like.

Example:

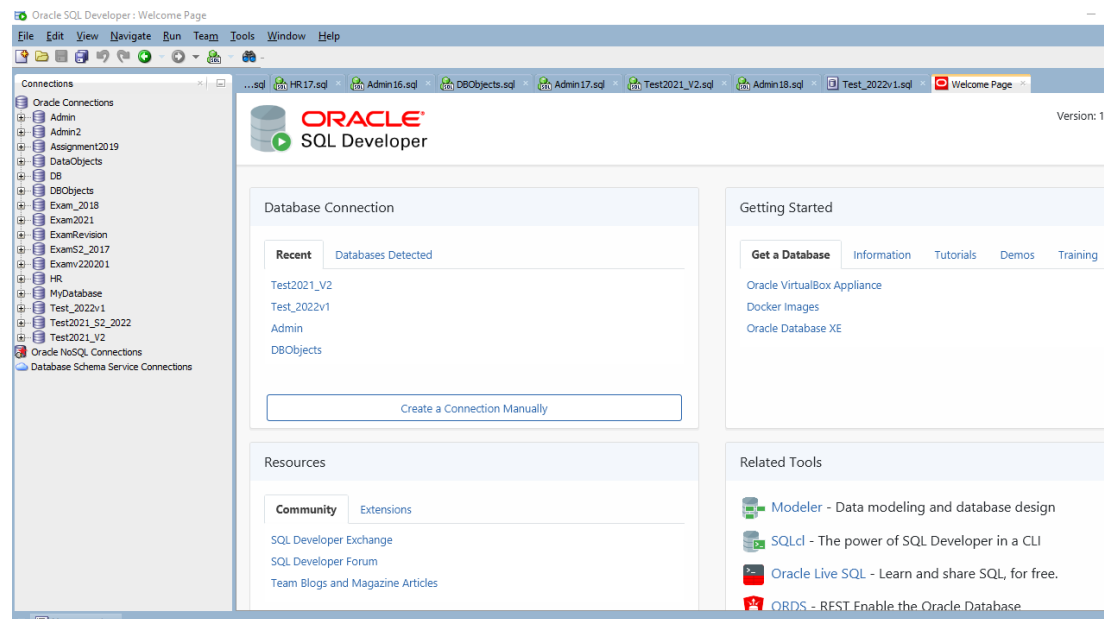


2. **SQL Developer:** Primarily we will be using Oracle SQL Developer for this course. Oracle SQL Developer is a fully supported graphical tool for database development. It is used to browse database objects, run SQL statements and scripts, edit, and debug PL/SQL statements. SQL Developer can connect to any Oracle database version. It is compatible with Windows, Linux, and Mac OS X.

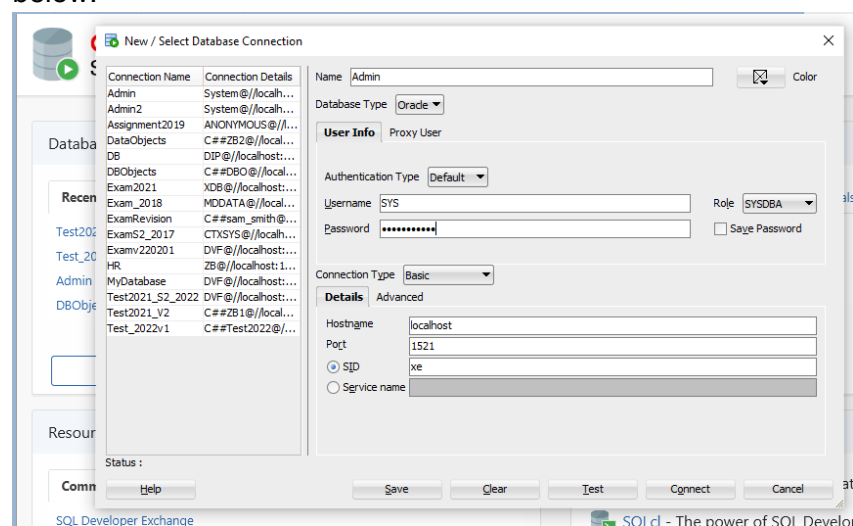
Oracle SQL Developer is a stand-alone application and is downloaded and installed separately from the Oracle database. You open it from the folder you downloaded it from. See screenshot below.

modules	2020/02/03 20:15	File folder	
netbeans	2020/02/03 20:15	File folder	
orakafka	2020/02/03 20:15	File folder	
rdms	2020/02/03 20:15	File folder	
sleepycat	2020/02/03 20:15	File folder	
sqldeveloper	2020/02/03 20:15	File folder	
sqlj	2020/02/03 20:15	File folder	
svkit	2020/02/03 20:15	File folder	
Desktop - Shortcut	2020/10/15 17:29	Shortcut	1 KB
icon	2019/12/20 17:59	PNG File	2 KB
sqldeveloper	2019/12/20 18:11	Application	89 KB
sqldeveloper	2019/12/20 17:59	Shell Script	1 KB

Once you double click on it you will come to this screen below

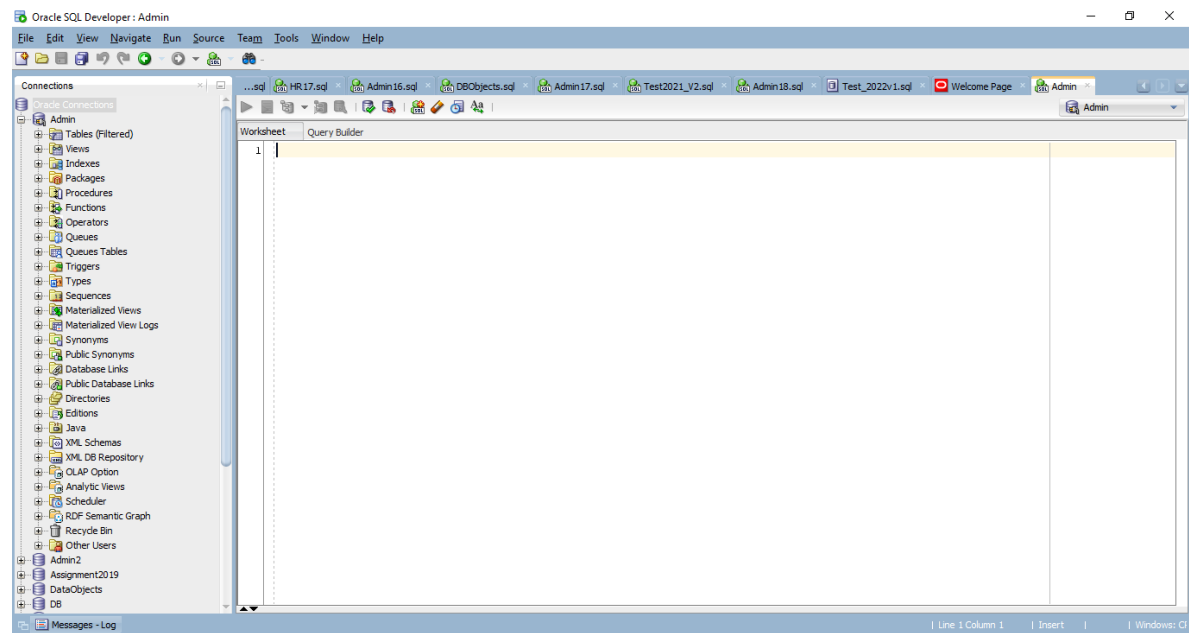


When you click on the green plus sign under connections you will come to the screen below.



If you are using the Virtual Machine the Username will be SYS and choose the SYSDBA role. The password will be on the readme file on the virtual machine.

You should then see the screen below where you can now start creating your objects.

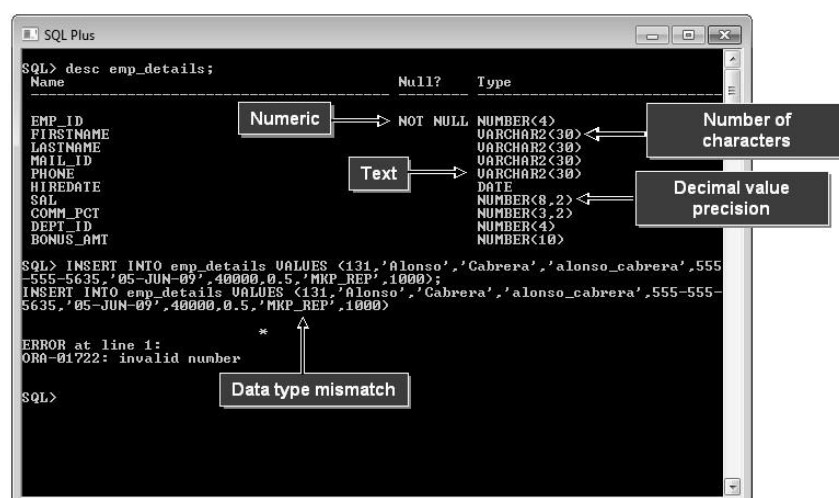


2.16 Data Types

Definition: Data Types

A data type is a classification of data into groups based on its characteristics. Characteristics may include the number of characters that a value can contain, whether the value is text or numeric, whether it includes decimals, and how it represents date and time values. In addition, a data type also restricts the column size, and determines the operations that can be performed on data. It's important that you ensure that the data type of the value entered into a column matches the data type assigned to that column. If there is a data type mismatch, the SQL server will return an error. Thus, every value that is entered into a SQL database is classified into one of the available data types (R, 2015).

Example:



Below are some of the commonly used data types as of my last knowledge update in September 2021:

Data Type	Description	Syntax
NUMBER	Represents fixed or floating-point numbers.	NUMBER(precision, scale), where precision is the total number of digits, and scale is the number of digits to the right of the decimal point
VARCHAR2	Variable-length character data.	VARCHAR2(size), where size is the maximum length of the string in characters
CHAR	Fixed-length character data	CHAR(size), where size is the fixed length of the string in characters.
DATE	Represents a date and time.	DATE.
TIMESTAMP	Stores date and time to fractional seconds	TIMESTAMP
CLOB	Large Character Object, for storing large text data.	CLOB
BLOB	Binary Large Object, for storing binary data (e.g., images, documents).	BLOB
BINARY_DOUBLE	Double-precision floating-point number.	BINARY_DOUBLE
BINARY_FLOAT	Single-precision floating-point number.	BINARY_FLOAT
RAW	Stores binary data as a series of bytes.	RAW(size) where size is the number of bytes.
BOOLEAN	Represents true or false values.	BOOLEAN.
INTEGER	Represents a 38-digit signed integer	INTEGER.
SMALLINT	Represents a 38-digit signed integer	SMALLINT
LONG	For handling variable-length character data, deprecated in favor of CLOB.	LONG.

NCHAR	Fixed-length character data using the national character set.	NCHAR(size).
NVARCHAR2	Variable-length character data using the national character set	NVARCHAR2(size).
NCLOB	Large Character Object using the national character set.	NCLOB.
INTERVAL	Stores intervals of time.	INTERVAL.
XMLTYPE	Stores XML data.	XMLTYPE
ROWID	Represents the unique address of a row in a table.	ROWID

2.17 Creating Tables

To create tables and insert records in Oracle, you'll need to use SQL commands. Below is an example of how to create tables and insert records using Oracle SQL. I'll use a simple example with two tables: one for customers and another for orders.

First, create a table for customers:

-- Create a table for customers

```
CREATE TABLE customers (  

      customer_id NUMBER(5) PRIMARY KEY,  

      first_name VARCHAR2(50),  

      last_name VARCHAR2(50),  

      email VARCHAR2(100),  

      phone VARCHAR2(20)  

);
```

This SQL statement creates a table named customers with columns for CustomerID, first_name, last_name, email, and phone. The customer_id column is defined as the primary key.

Next, create a table for orders and establish a foreign key relationship with the customers table:

-- Create a table for orders with a foreign key to the customers table:

```
CREATE TABLE orders (  

      order_id NUMBER(10) PRIMARY KEY,
```

```
order_date DATE,  
total_amount NUMBER(10, 2),  
customer_id NUMBER(5),  
FOREIGN KEY (customer_id) REFERENCES customers (customer_id)  
);
```

This SQL statement creates a table named `orders` with columns for `order_id`, `order_date`, `total_amount`, and `customer_id`. The `customer_id` column is a foreign key that references the `customer_id` column in the `customers` table.

Now, you can insert records into these tables. Here's an example of how to insert records into the `customers` and `orders` tables:

-- Insert records into the customers table

```
INSERT INTO customers (customer_id, first_name, last_name, email, phone)  
VALUES (1, 'John', 'Doe', 'john.doe@example.com', '555-123-4567');
```

```
INSERT INTO customers (customer_id, first_name, last_name, email, phone)  
VALUES (2, 'Jane', 'Smith', 'jane.smith@example.com', '555-987-6543');
```

-- Insert records into the orders table

```
INSERT INTO orders (order_id, order_date, total_amount, customer_id)  
VALUES (1001, TO_DATE('2023-10-12', 'YYYY-MM-DD'), 150.00, 1);
```

```
INSERT INTO orders (order_id, order_date, total_amount, customer_id)  
VALUES (1002, TO_DATE('2023-10-13', 'YYYY-MM-DD'), 200.50, 2);
```

These SQL INSERT statements add records to the `customers` and `orders` tables. The `TO_DATE` function is used to insert dates in the specified format.

After executing these SQL commands, you will have created tables for `customers` and `orders` and inserted records into them in your Oracle database.

Let's say you want to add a new column to the `customers` table created earlier. You can use the `ALTER TABLE` statement to make this change. Here's an example of how to add a new column called `address` to the `customers` table:

-- Add a new column to the customers table

```
ALTER TABLE customers  
ADD address VARCHAR2(100);
```

This `ALTER TABLE` statement modifies the `customers` table by adding a new column called `address` with a data type of `VARCHAR2(100)`.

After executing this statement, the `customers` table will have the new `address` column, and you can start inserting or updating data in this column.

If you want to change the data type of the address column in the customers table, you can use the ALTER TABLE statement with the MODIFY clause. Here's an example of how to change the data type of the address column from VARCHAR2(100) to CLOB:

-- Change the data type of the address column in the customers table

```
ALTER TABLE customers  
MODIFY address CLOB;
```

This ALTER TABLE statement modifies the customers table by changing the data type of the address column to CLOB. This is useful when you need to store large amounts of text data in the address column, as CLOB allows for much larger text content compared to VARCHAR2.

After executing this statement, the address column will have the data type CLOB.

2.18 The SELECT Statement

SELECT is a SQL statement that is used to retrieve information from a database. The SELECT statement is composed of two parts—a SELECT clause that includes all required column names in the output, and a FROM clause that contains the table name and columns. More clauses can be added to the SELECT statement, if necessary. The order of column names used in the SELECT statement determines the order in which the columns are displayed in the output. Commas are used to separate column names when more than one column name is entered.

When all columns of a table are required in the output, an asterisk (*) can be used instead of column names.

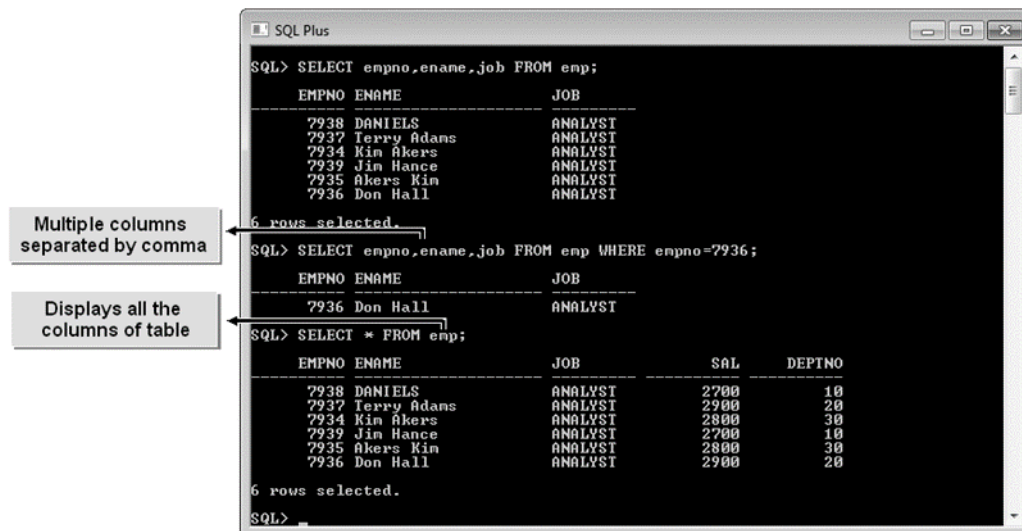
The syntax of the SELECT statement is

```
SELECT colname1[, colname2, colname3 ...]  
FROM tablename
```

In the syntax, the square parentheses [...] are used to enclose optional parameters.

To retrieve all columns from a table, you can use:

```
SELECT *  
FROM tablename
```

Comparison Operators Used with the SELECT Command The table below provides you with a list of comparison operators and their usage with the SELECT command.

Condition	Use	Example
$A = B$	Tests equality. Returns TRUE if A and B are equal.	<code>SELECT * FROM emp WHERE dept_id = 20;</code>
$A \neq B$, $A <> B$, $A \neq B$	Tests inequality. Returns TRUE if A and B are not equal.	<code>SELECT * FROM emp WHERE dept_id != 20;</code>
$A > B$	Tests for a range of values. Returns TRUE if A is greater than B.	<code>SELECT * FROM emp WHERE dept_id > 20;</code>

Condition	Use	Example
$A < B$	Tests for a range of values. Returns TRUE if A is less than B.	<code>SELECT * FROM emp WHERE dept_id < 20;</code>
$A \geq B$	Tests for a range of values. Returns TRUE if A is greater than or equal to B.	<code>SELECT * FROM emp WHERE dept_id >= 20;</code>
$A \leq B$	Tests for a range of values. Returns TRUE if A is less than or equal to B.	<code>SELECT * FROM emp WHERE dept_id <= 20;</code>

Optional Clauses of the SELECT Statement

The optional clauses of the SELECT statement, if used, must be in a specific order. The optional clauses serve their own purposes (R, 2015).

Optional Clause	Purpose
WHERE	You can use this clause to specify a condition to retrieve only certain rows from a table.
GROUP BY	You can use this clause to organize data into groups.

Optional Clause	Purpose
HAVING	You can use this clause to specify a condition that works in conjunction with GROUP BY, specifying the groups to be included in the results.
ORDER BY	You can use this clause to specify a condition that can sort query results by one or more columns.

2.19 Aliases

Definition:

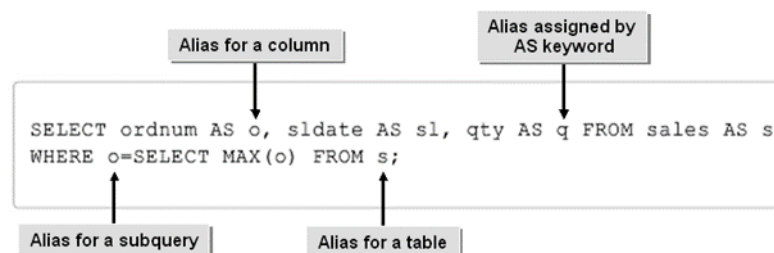
An alias is a temporary name given to a table, column, or subquery with a complex or long name. The alias can be assigned using the AS keyword. The AS keyword for the column alias is optional; however, the AS keyword that appears immediately after the tablespace name in the CREATE clause is required when creating a table based on a subquery(R, 2015).

The syntax for aliasing a column is:

```
SELECT column_name AS
alias_name
FROM table_name;
```

The syntax for aliasing a table is:

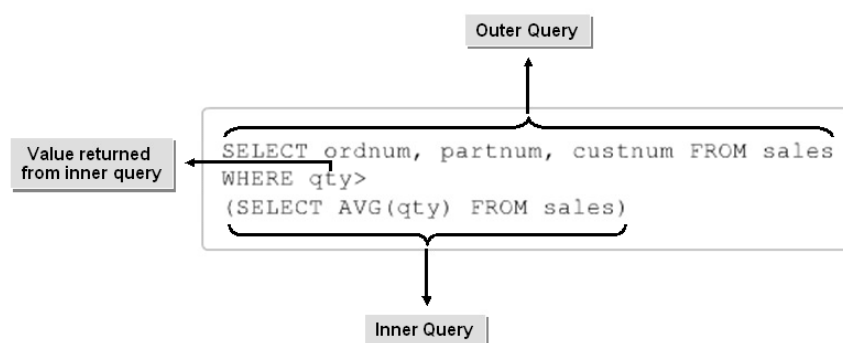
```
SELECT column_name(s)
FROM table_name
AS alias_name;
```



2.20 Subqueries

Definition:

A subquery, also known as an inner query, is a query that is nested within another query. The query that contains the subquery is known as the outer query. A subquery is used when the value needed by the outer query's condition is unknown. The subquery retrieves the value and returns it to the outer query. It can also return computed values. Thus, the inner query is always executed before the outer query. As the number of inner queries may be one or more, multiple inner queries are separated by parentheses. Inner queries enable you to sort and group records retrieved by the subquery(R, 2015).



The outer query returns data based on specific condition and the inner query returns the value to the outer query

2.21 The Group By Function

Group By is a keyword used to select multiple columns from a table such that there appears at least one arithmetic operation in the SELECT statement. It helps you to perform functions on entire columns of data in a single query(R, 2015).

The Group by supports various functions.

AVG ()	Calculates an average on a column.
COUNT ()	Counts the number of items in a column excluding null values.
MAX ()	Determines the maximum value in a column.
MIN ()	Determines the minimum value in a column.
SUM ()	Calculates a sum on a column.

The Having Clause

You may need to filter or limit the result set after the initial data is grouped and aggregated. This clause is typically placed near the end of the SQL statement, which may or may not include the GROUP BY clause. The syntax for this clause is :

```
SELECT "column_name1", SUM("column_name2")  
FROM "table_name"  
GROUP BY "column_name1"  
HAVING (arithmetic function condition)
```

2.22 JOINS

Definition:

A join is a query that is used to combine values in two or more tables in a relational database. It results in a temporary table called the joined table. A join connects tables by using their key fields.

Example:

You have two tables Employees and Projects. The Employees table has the EmployeeName and ProjectID fields, and the Projects table has the ProjectID and ProjectName fields. Some but not all of the ProjectIDs in the Employees table are in the Projects table, and some of the ProjectIDs appearing in the Projects table are not listed in the Employees table. To analyse which employee is assigned to a project and what the projects are that have no employees assigned, you need to combine the information contained in these tables by using the join query.

EmployeeName	ProjectID
Amy	4521
John	9658
George	9658
Melissa	5230
Andrew	4521
Chris	8874

ProjectID	ProjectName
4521	Star
9658	Zenith
5230	Symphony
1125	Genesis

EmployeeName	ProjectID	ProjectName
Amy	4521	Star
John	9658	Zenith
George	9658	Zenith
Melissa	5230	Symphony
Andrew	4521	Star
Chris	8874	
	1125	Genesis

Joined Table

```
SELECT EmployeeName, ProjectID, ProjectName FROM Employees  
INNER JOIN Projects ON Employees.ProjectID = Projects.ProjectID
```

The query joins the Employees and Projects tables based on the ProjectID column. In Oracle Joins are easier as you can use Natural Joins. For example the above query can be done as follows:

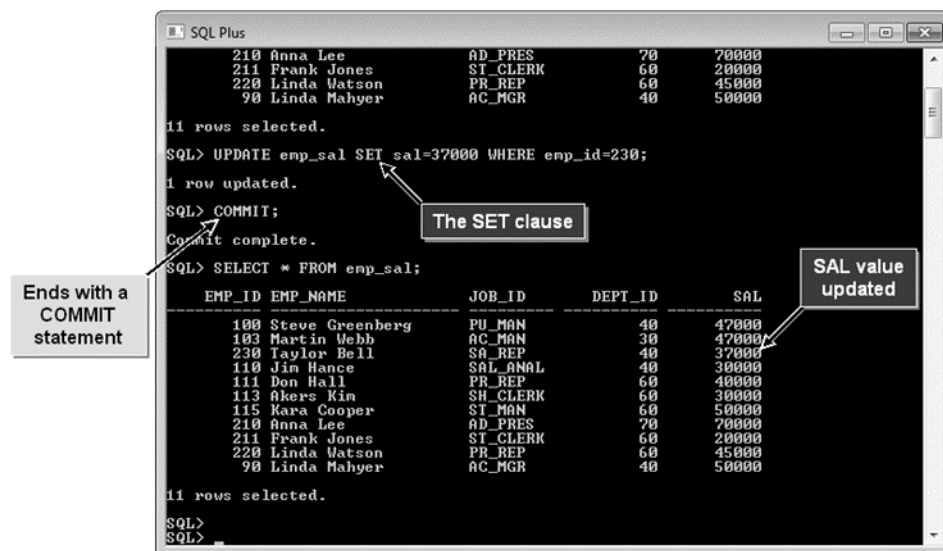
```
SELECT EmployeeName, ProjectID, ProjectName from Employees NATURAL  
JOIN Projects.
```

2.23 The UPDATE Statement

The UPDATE statement is used to modify existing data in a database. The SET clause is used to assign new values to the columns of a table. Like other DML statements, this statement also uses the COMMIT statement at the end of the query to make permanent changes to data and prevent the data from rolling back in case of any network failure(R, 2015).

The syntax for this statement is:

UPDATE *table_name*
SET *column_name* = *value* / *expr* / *subquery* [...]
[WHERE condition];
COMMIT;

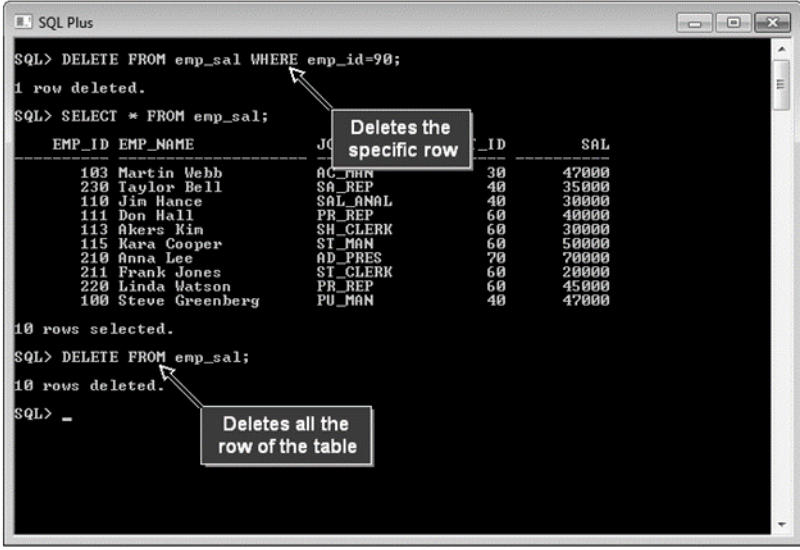


2.24 The DELETE Statement

The DELETE statement is used to remove data from a table. The only clause used in the DELETE statement is WHERE, and it is optional. But if this clause is not used, all rows will be removed from the table(R, 2015).

The syntax for the DELETE statement is:

DELETE FROM *table_name*
[WHERE condition];



The screenshot shows a SQL Plus window with the following text:

```
SQL> DELETE FROM emp_sal WHERE emp_id=90;
1 row deleted.
SQL> SELECT * FROM emp_sal;
```

A callout box labeled "Deletes the specific row" points to the WHERE clause in the first DELETE statement.

EMP_ID	EMP_NAME	JOB	DEPT	MANAGER	START_DATE	END_DATE	SAL
103	Martin Webb	AC_FRN	30				47000
230	Taylor Bell	SA_REP	40				35000
110	Jim Hance	SAL_ANAL	40				30000
111	Don Hall	PR_REP	60				40000
113	Akers Kim	SH_CLERK	60				30000
115	Kara Cooper	ST_MAN	60				50000
210	Anna Lee	AD_PRES	70				70000
211	Frank Jones	ST_CLERK	60				20000
220	Linda Watson	PR_REP	60				45000
100	Steve Greenberg	PU_MAN	40				47000

10 rows selected.

```
SQL> DELETE FROM emp_sal;
10 rows deleted.
SQL> _
```

A callout box labeled "Deletes all the row of the table" points to the DELETE FROM statement.

2.25 *Use Transaction Control Language (TCL) Commands*

In the last topic, you manipulated data in the table. These changes need to be controlled to manage the database effectively. In this topic, you will use Transaction Control Language (TCL) commands.

You are able to manipulate data using DML commands. Have you ever thought of a situation where transactions or changes should be retained permanently or temporarily? Also, what if you change data inadvertently? Could you undo the changes if you had to? The skills you'll learn in this topic will enable you to control data transactions like this by using TCL commands(R, 2015).

TCL

Transactions, which are a group of actions performed in a logical order, are controlled using the Transaction Control Language (TCL). TCL consists of statements or queries that help manage the changes made to database objects. It is also used to group DML statements into transactions. You also use TCL for activities like saving changes or preventing changes from being made to a database. Examples of commands in TCL are commit, roll back, save point and set transaction(R, 2015).

The COMMIT Statement

You can make permanent changes to data by using the COMMIT statement. When you issue this statement, it ends the current transaction and makes the changes permanent. These changes are not visible to other users until the COMMIT statement is issued.

Suppose you are changing the salary of an employee named Scott from 3000 to 3500. For other users, the salary of Scott will be displayed as 3000 and for you it will be 3500. Once the COMMIT statement is issued, the updated salary 3500 will be displayed for other users as well(R, 2015).

The syntax for the COMMIT statement is:

***INSERT INTO table_name (column_name [...]) VALUES
(value | expr [...]); COMMIT;***

The screenshot shows a SQL Plus window with the following commands and output:

```

SQL> INSERT INTO emp_sal VALUES (204,'Nancy Hawkins','ST_MAN',60,34000);
1 row created.
SQL> UPDATE emp_sal SET dept_id=40 WHERE emp_id=100;
1 row updated.
SQL> DELETE FROM emp_sal WHERE emp_id=210;
1 row deleted.
SQL> COMMIT;
Commit complete.
SQL> SELECT * FROM emp_sal;

```

Annotations in the image:

- A box labeled "Ends the transaction" with an arrow pointing to the `COMMIT;` command.
- A box labeled "Table update as per transaction" with an arrow pointing to the `SELECT * FROM emp_sal;` query.

The result of the `SELECT` query is shown as a table:

EMP_ID	EMP_NAME	JOB_ID	DEPT_ID	SAL
204	Nancy Hawkins	ST_MAN	60	34000
103	Martin Webb	AC_MAN	30	47000
230	Taylor Bell	SA_REP	40	35000
110	Jim Hance	SAL_MAN	40	30000
111	Don Hall	PR_REP	60	46000
113	Akers Kim	SH_CLERK	60	30000
115	Kara Cooper	ST_MAN	60	50000
211	Frank Jones	ST_CLERK	60	20000
220	Linda Watson	PR_REP	60	45000
90	Linda Mahyer	AC_MGR	40	50000
100	Steve Greenberg	PU_MAN	40	47000

11 rows selected.

The Rollback Segment

By using the ROLLBACK statement, you can undo the changes made in the current transaction. You can also manually undo the changes made in an in-doubt distributed transaction. Oracle always recommends that you explicitly commit or roll back the changes made in the current transaction (R, 2015). Otherwise the program terminates abnormally and the last transaction changes are rolled back automatically.

The syntax for the ROLLBACK statement is:

**INSERT INTO table_name (column_name [,...]) VALUES
(value | expr [,...]); ROLLBACK;**

You do not require any privileges to roll back transaction changes. However, you need certain privileges to roll back changes manually. You need the FORCE TRANSACTION system privilege to manually roll back your committed changes, and the FORCE ANY TRANSACTION system privilege to manually roll back the changes committed by other users (R, 2015).

The Rollback Segment

The rollback segment is an area used to hold the before-image of data that is undergoing some modification by a DML statement. A before-image is a snapshot of the original, unmodified data. All DML statements, such as INSERT, UPDATE, and DELETE, will generate a before- image in rollback segments.

This before-image is used for three main purposes:

Read-consistency, to query a portion of data that is under modification but not yet committed.

Recovery, to roll back any transactions that are in an uncommitted state at the point of a system failure.

Rollback to undo a previous action

OPTIMAL Specifies the size that the rollback segment should shrink back after all transactions that were using the segment are complete.

The syntax of the CREATE ROLLBACK SEGMENT statement is

```
CREATE ROLLBACK SEGMENT segment_name
.....STORAGE (
.....INITIAL n [K|M]

.....NEXT n [K|M]
.....MINEXTENTS n
.....MAXEXTENTS n
.....OPTIMAL n [K|M]
.      );
```

where,

- **CREATE ROLLBACK SEGMENT**: Keyword to create the rollback segment.
- **INITIAL**: Storage attribute that specifies the size of the first extent of the rollback segment.
- **NEXT**: Attribute that specifies the size of all subsequent extents of the segment.
- **MINEXTENTS**: Specifies the minimum number of extents for the segment, where the minimum and default value is two.
- **MAXEXTENTS**: Specifies the maximum number of extents for the segment, and its default value is UNLIMITED.
- **OPTIMAL**: Specifies the size that the rollback segment should shrink back to after all transactions that were using the segment are complete.

The SAVEPOINT Statement

A SAVEPOINT is a marker used to identify a specific transaction point. You can create this SAVEPOINT using the SAVEPOINT statement. The names given for the SAVEPOINT should be unique each time you create names. If names are identical, then the latest SAVEPOINT overwrites the previous one. The main purpose of creating a SAVEPOINT is that you can roll back to that specific transaction point later. Issuing a ROLLBACK command at the end will undo the changes made in the entire transaction and also deletes all savepoints(R, 2015).

The SAVEPOINT Statement

Suppose you insert a row into an employee table and name the transaction as emp_insert. You then delete a row from the employee table and name the transaction as emp_del. You update one of the employees' salary and name this transaction as emp_update. Now, if you issue a ROLLBACK statement, it will undo all the transactions that are made. But if you issue the command ROLLBACK emp_insert, it will undo only that change retaining the other changes(R, 2015).

```

SQL> UPDATE emp_duration SET end_date='06-JAN-05' WHERE emp_id=111;
1 row updated.
SQL> SAVEPOINT emp_up;
Savepoint created.
SQL> DELETE FROM emp_duration WHERE emp_id=155;
1 row deleted.
SQL> SAVEPOINT emp_del;
Savepoint created.
SQL> ROLLBACK TO emp_up;
Rollback complete.
SQL> _

```

Rollback complete.

Rolled back to a specific transaction

2.26 The SET TRANSACTION

The SET TRANSACTION Statement

The SET TRANSACTION statement is used to set the current transaction as read-only or read- write. You can also set an isolation level for the transaction or assign the transaction to a rollback segment. The changes made in the SET TRANSACTION section affect only the current transaction of yours and not other users or other transactions. However, the transaction ends when a COMMIT or a ROLLBACK is issued (R, 2015).

The syntax of the SET TRANSACTION statement is:

SET TRANSACTION READ ONLY/READ WRITE NAME name of the transaction ISOLATION LEVEL setting ROLLBACK SEGMENT;

```

SQL Plus
Copyright (c) 1982, 2010, Oracle. All rights reserved.
Enter user-name: mcalla@orcl
Enter password:
Connected to:
Oracle Database 11g Release 11.1.0.6.0 - Production
SQL> SET TRANSACTION READ ONLY NAME 'Employee Name';
Transaction set.
SQL> SELECT emp_id, firstname, lastname FROM emp_details WHERE emp_id=157;
EMP_ID FIRSTNAME LASTNAME
-----
157 Laura Murray
SQL> COMMIT;
Commit complete.
SQL> SET TRANSACTION READ WRITE NAME 'Employee Bonus';
Transaction set.
SQL> UPDATE emp_details SET bonus_amt=4000 WHERE emp_id=130;
1 row updated.
SQL> COMMIT;
Commit complete.

```

Read only transaction

Read write transaction

In the Figure above, The select and update transactions are set to READ ONLY and READ WRITE respectively.

How to Use Transaction Control Language (TCL) Commands

Procedure Reference: Control a Transaction

1. Insert, delete, or update data using DDL or DML commands.
2. Enter commit; to commit all transactions made.
3. To save the transaction made, enter savepoint name; where name can be any desired name.
4. Undo changes made in the transaction.
 - To undo the changes made in all transactions, enter rollback;
 - To undo the changes made in a specific transaction, enter rollback savepoint name.

3 Revision Exercises

1. A database is divided into layers or tiers to manage data effectively. Briefly explain the differences between the two database architecture tier types.
2. Mike is a database administrator who has recently joined an online car rental company who captured and stored data in Microsoft Excel spreadsheets. He is tasked with creating a database and migrating all the data into a relational database management system. Mike wants to implement an Oracle database, but is unsure of the Oracle database edition to use. Explain to Mike the various program categories that will enable him to operate a database in Oracle. Discuss at least three of these program categories.

4 Practical Activities

The following set of relations has been set up for a local private investment organisation. At present the database is small and only includes information about brokers, customers and transactions. The relationships between the tables must be derived from the data in each of the tables. The tables and the information required are as follows:

BROKER (BROKER_ID, FIRST_NAME, SURNAME, EMAIL)
CUSTOMER (CUSTOMER_ID, FIRST_NAME, SURNAME, EMAIL)
TRANSACTIONS (TRANS_ID, TRANS_TYPE, TRANS_DATE, TRANS_AMT,
BROKER_ID, CUSTOMER_ID)

Sample Data is shown below:

BROKER

BROKER_ID	FIRST_NAME	SURNAME	EMAIL
111	Lucy	Jackson	lack@yahoo.com
112	Molly	Jones	jonesm@gmail.com
113	Andrea	Paulson	ap@gmail.com

CUSTOMER

CUSTOMER_ID	FIRST_NAME	SURNAME	EMAIL
1001	Jack	Smith	js@yahoo.com
1002	Sam	Hewson	shew@gmail.com
1003	Alberto	Goodwin	good@isat.co.za

TRANSACTIONS

TRANS_ID	TRANS_TYPE	TRANS_DATE	TRANS_AMT	BROKER_ID	CUSTOMER_ID
Trans_101	Pension Pay-out	15-MAR-19	275 000	112	1001
Trans_102	Bitcoin Purchase	17-MAR-19	180 000	113	1003
Trans_103	Kruger Rands Purchased	22-MAR-19	150 000	113	1003
Trans_104	Mortgage Deposit	25-MAR-19	35 000	111	1002

1. You will need to create the above tables to complete the test. Please create the tables and populate them using SQL Developer or SQL*Plus.
2. Create a SQL Query to display the broker name and how many transactions they were involved in. Sample output shown below:

Sample Results

BROKER_NAME	PURCHASE_COUNT
Andrea Paulson	2
Molly Jones	1
Lucy Jackson	1

3. Create a SQL query to display the broker ID, customer ID and the transaction amount. In your query include the 10% commission value and the total transaction amount. Sample output shown below:

Sample Results

BROKER_ID	CUSTOMER_ID	TRANS_AMT	COMMISSION	TOTAL
112	1001	R275 000	R27 500	R302 500
113	1003	R180 000	R18 000	R198 000
113	1003	R150 000	R15 000	R165 000
111	1002	R35 000	R3 500	R38 500

Learning Unit 2: Administering a Database

Learning objectives addressed in this learning unit:

- LO1: Explain how to create a database.
- LO2: Explain the database configuration assistant (DBCA).
- LO3: Describe the various administrator tasks and privileges to perform database operations.
- LO4: Describe the initialisation parameters to accomplish automatic memory management.
- LO5: Compare various methods associated with manual memory management.
- LO6: Describe how to manage database memory.
- LO7: Compare the types of databases server processes.
- LO8: Describe how to manage processes effectively.
- LO9: Explain how to use table spaces to optimise table storage space.
- LO10: Explain predefined and administrative user accounts.
- LO11: Explain how to assign and revoke user privileges.

My notes

1 Introduction

You have managed database operations by using SQL and PL/SQL commands. The various database objects are accessed by several users. Now you may want to perform some basic database administration tasks. In this lesson, you will administer a database. Is it enough that you just have a rudimentary knowledge on the design and working of an Oracle database? As a database administrator, you may have to administer a database by creating a new database or customizing an existing one on a real-time basis. You should also manage and monitor database components and processes regularly. So, how might you do all this? You may have to perform these tasks in various stages of database creation and maintenance(R, 2015).

2 Create a Database

You manipulated data in a database using PL/SQL commands. As an organization grows, so does data it needs to manage. As a DBA, you may require new databases to group data logically. In this topic, you will create a database.

As a DBA, you will need to manage the database and its components. As your organization grows, and business requirements change, there will always be a need to create new databases. Other than the existing databases, you will be creating new ones more frequently. The CREATE DATABASE statement is a SQL command that is used to create a database with a unique name in a specific server environment. The new database remains inaccessible to users until the DBA or the creator of the database grants them the permission to access it (R, 2015).

Once the database is created, it can hold various SQL objects such as tables, fields, rows, views, and indexes.

The syntax for creating a database is:
CREATE DATABASE database_name

2.1 DBCA

The Database Configuration Assistant (DBCA) provides a simple, point-and-click interface to assist users in creating a database. It includes most aspects of database creation, such as file names and placement, and basic initialization parameter settings. A Java-based program, DBCA, runs either as a stand-alone application or as part of the Oracle Universal Installer (OUI). It allows you to assign memory options, file locations, database size, storage parameters, or parameters in the init.ora file, and SGA size, and also locate administrative files. Further, it is used for transaction processing and data warehousing (R, 2015).

The two memory options available by DBCA are typical and custom. The typical memory options provide the default options that allow minimum user interaction. The custom memory options are used by advanced users to alter and customize memory settings. DBCA also defines the maximum amount of memory to be assigned to a database (R, 2015).

DBCA Templates

DBCA templates are used to create a database or duplicate the existing one; they exist in the XML format and may be used with datafiles. The templates store the details of database options, initialization parameters, and storage attributes (R, 2015).

2.2 Database Administrator Tasks

The DBA plays the most important and critical role in database management. The functions of a DBA include planning, creating, managing, and maintaining a database (R, 2015). The DBA administers the most critical database tasks by:

- Installing and maintaining Oracle software.
- Initiating and restoring the Database Environment (DBE).
- Accepting or rejecting changes in the schema and database design.
- Creating and controlling the Oracle schema object.
- Managing all logs such as redo, SQL, and the buffers related to them.
- Managing the transactions, locks, and performing database backup and recovery.
- Manage data security.
- Monitoring the database, tuning its performance, and implementing preventive measures to avoid system crash.
- Upgrading the database with new releases.
- Analysing the future requirements for the database.

2.3 Administrative Privileges

An Oracle DBA is provided with administrative privileges to perform database operations. These privileges are granted using two system privileges: SYSDBA and SYSOPER. These system privileges are controlled outside the database, thus enabling the DBA to access the database instance even when the database is closed. While the connection to the SYSDBA privilege is enabled using the default schema, SYS, the connection to the SYSOPER privilege is provided using PUBLIC.

The SYSOPER privilege allows you to perform basic operational tasks without looking at user data, but the SYSDBA privilege enables a user to connect as SYS and monitors users' data.

Both SYSOPER and SYSDBA allow you to:

- Perform database startup and shutdown operation.
- Create the SPFILE or Server Parameter file.
- Alter a database to open, mount, or perform backup.
- Alter the database archive log.
- Enable complete recovery of the database.
- Provide the RESTRICTED SESSION privilege.
- In addition, SYSDBA allows you to:
- Connect as user SYS.
- Create or drop databases.
- Perform complete or incomplete recovery of a database. Incomplete recovery includes the TIME \ CHANGE \ CANCEL \ CONTROL FILE recovery.

2.3.1 How to Create a Database Using DBCA

To create a database in Oracle SQL Developer, you typically use the Database Configuration Assistant (DBCA), a tool provided by Oracle to simplify the database creation process . Here are the steps to create a database using DBCA:

Launch Oracle SQL Developer:

Start by opening Oracle SQL Developer on your computer.

Connect to a Database:

Before creating a new database, you need to connect to an existing Oracle database. If you don't have an existing database, you can connect to a built-in database like XE or a remote database.

Start DBCA:

Once you are connected to a database, go to the "Tools" menu and select "Database Configuration Assistant" (DBCA).

Select "Create a Database":

In the DBCA wizard, choose the "Create a database" option and click "Next".

Database Templates:

You can choose from various database templates provided by Oracle. These templates pre-configure the database for specific purposes, such as a general-purpose database, a data warehouse, or an OLAP database. Select the template that suits your requirements and click "Next".

System Class and Database Identifiers:

Here, you need to specify the system class, which typically corresponds to the size and capabilities of your server, and the database identifier (DB_UNIQUE_NAME). Configure these settings and click "Next".

Configure Database Options:

In this step, you can configure various options for your database, such as the character set, the total database size, and the memory allocation for the database. Customize these settings according to your needs and click "Next".

Management Options:

Choose whether you want to configure your database for automatic memory management and automatic undo management. Make your selections and click "Next".

Configure Database Storage:

Define the storage options for your database, including data file locations, control file locations, and redo log file locations. Customize these settings and click "Next".

Database File Locations:

Set the location for your database files, including data files, control files, and redo log files. You can use the default locations or specify custom paths. Click "Next" when you're done.

Recovery Configuration:

Configure backup and recovery options for your database. You can choose to use Recovery Manager (RMAN) or opt for manual backups. Make your selections and click "Next".

Initialization Parameters:

You can set additional initialization parameters for your database if needed. Customize these settings, or use the default values, and click "Next".

Database Content:

Specify the type of data you want to include in your database, such as sample schemas, Enterprise Manager data, or custom scripts. Make your selections and click "Next".

Summary:

Review the summary of your database configuration settings. If everything looks correct, click "Finish" to begin the database creation process.

Database Creation Progress:

DBCA will now create the database based on the configuration you specified. You can monitor the progress in the DBCA interface.

Database Created:

Once the database creation is complete, you will receive a confirmation message. Click "OK" to finish the process.

Your Oracle database is now created using DBCA. You can connect to your newly created database using Oracle SQL Developer and start working with it.

2.3.2 Manage Database Memory

You have created a database using DBCA. After you create a database, you may need to manage the memory space accordingly so as to effectively run the newly created database. In this topic, you will manage database memory.

You may create databases as per business requirements. What will happen if you create databases over and over again to match up changing requirements? What will be the impact on available memory? As a DBA, it is a primary requirement that you check memory allocation and support resourceful utilization of the available memory so as to ensure high database performance (R, 2015).

Automatic Memory Management

You can simplify and automate memory management in Oracle by automatically allocating space for both the System Global Area and the Program Global Area. You can manage the SGA and PGA with a single point of control (R, 2015).

To accomplish automated memory management, Oracle uses two initialization parameters.

MEMORY_MAX_TARGET This parameter specifies the maximum amount of memory that the Oracle instance can use for both the System Global Area (SGA) and Program Global Area (PGA) combined. It represents an upper limit on the total memory allocation to the Oracle instance.

MEMORY_TARGET Determines the memory available for the instance and allows the instance to allocate this memory to either SGA or PGA.

Manual Memory Management

Manual memory management enables you to have better control over the memory size of individual components. Memory can be managed manually by disabling the automatic memory management option. Memory can be managed using various methods that include the retention of automation property (R, 2015).

Automatic PGA Memory Management

Soon after the target size of the instance PGA is estimated, the amount of memory that an instance can use is determined easily. Oracle then dynamically controls the size of the individual PGA. The target size of instance PGA is set explicitly in order to enable

you to compute and configure the size according to the requirement rather than using the default size (R, 2015).

Manual PGA Memory Management

You can manage the PGA memory manually by disabling the automatic PGA memory management option. It enables you to set the work area size for each SQL operator. But, because the database workload keeps on changing very frequently, this method of memory management is not very often used (R, 2015).

How to Manage Database Memory

Procedure Reference: Manage Memory Using the Enterprise Manager

To manage memory using the Enterprise Manager:

1. Choose Start ▢ All Programs ▢ Oracle— OraDb11g_home1 ▢ Database Control
2. In the browser window, in the User Name text box, enter user name. Only users with administrative rights over the database can login to manage memory.
3. In the browser window, in the Password text box, enter the password.
4. From the Connect As drop-down list, select SYSDBA.
5. Click Login.
6. On the Database Instance: <database_name> page, click the Server link.
7. On the Database Instance: <database_name> page, in the Database Configuration section, click the Memory Advisors link.
8. If necessary, in the Memory Advisors section, click Disable to disable automatic memory management.
9. Set the SGA memory size.
 - a. Scroll down and in the Current Allocation section, in the Total SGA Size text box, enter the desired value to set the total SGA size.
 - b. In the Maximum SGA Size section, in the Maximum SGA Size text box, enter the desired value to set the maximum SGA size.
 - c. Click the Apply link
 - d. On the Advisor Central page, in the Confirmation section, click Yes to restart the database.
 - e. In the Host Credentials section, in the Username text box, type the host username.
 - f. In the Host Credentials section, in the Password text box, type the desired password.
 - g. In the Database Credentials section, in the Username text box, type the host username.
 - h. In the Database Credentials section, in the Password text box, type the desired password.
 - i. Click the OK link
 - j. On the Restart Database: Confirmation page, click the Yes link.
 - k. Click the Refresh link.
10. Set the PGA memory size.
 - a. Click the PGA link.
 - b. In the PGA section, in the Aggregate PGA Target text box, enter the desired value to set the aggregate PGA size.
 - c. Click the Apply link.

2.3.3 Manage Database Processes

You have set memory space to suit your needs. But the performance of the database also depends on the various processes running within the database application. In this topic, you will manage database processes (R, 2015).

What happens if several people try to access a single website at the same time? It will take an annoyingly long time to load a page if the website is not designed to handle heavy traffic.

Similarly, if multiple users access a database all at once, it slows down database performance, affecting the data retrieval process. So, it is critical that you manage processes robustly to enhance performance and efficiency (R, 2015).

2.4 Oracle Database Server Processes

The Oracle database server basically runs two types of processes: the user (client) process, and the Oracle process.

Process	Function
User (client) process	This process connects to the database using an interface such as SQL*Plus, and enables you to run the SQL codes, or run an Oracle tool such as Enterprise manager and Recovery manager.
Oracle process	Once users place their requests to Oracle the requests are managed by the Oracle application by creating server processes. The server process interacts both with the server and the user process to carry out the required operation. Depending on the type of server process that is dedicated or shared, the server process may handle requests for one or more user processes.

2.4.1 Dedicated Server Processes

A dedicated server process is confined to a single client connection such that each session in the process has an individual process. In this process, the user and the administrator are both connected to the instance explicitly for submitting a task, and this process uses the recovery manager to recover the database. Users connect to a database through the dedicated server by using a net service name that includes the SERVER=DEDICATED clause in the connect descriptor and provides the server with no idle time (R, 2015).

You can determine whether a session has a dedicated or shared process by using a query.

```
select sid, username, server from v$session where type='USER';
```

Shared Server Processes

A shared server configuration allows two or more users to share processes in order to lower the number of server processes and enhance the utility of available resources. In such a configuration, the user process is connected to the dispatcher, which supports multiple client connections. The user process is allotted a set of shared memory to place its requests and receive replies. Whenever a request arrives, the dispatcher places it in queue, and an idle shared server picks up the request to operate on it. It also provides you with the connection pooling option, which allows a session for a specific time; otherwise, it declares a time-out (R, 2015).

How to Manage Processes

Procedure Reference: Terminate a Session

To terminate a session:

1. Log in to the Enterprise Manager as a user with SYSDBA privileges.
2. On the Database Instance:<Database Name> page, click the Performance link.
3. In the Additional Monitoring Links section, click the Top Consumers link.
4. Click the Top Sessions link.
5. From the Select option, select the desired session.
6. Click the Kill Session link.
7. On the Confirmation page, click the Yes link.

2.4.2 Work with Tablespaces

You have created a database and managed the memory space based on requirement. As the volume of data increases, you need to manage storage space efficiently to ensure efficient data processing and retrieval. Therefore, it is critical that steps are taken regularly to manage and enhance data storage. In this topic, you will work with tablespaces to optimize table storage space (R, 2015).

While administering a database, you need to keep track of storage space for efficient data processing and retrieval. So, you will be monitoring the percentage of used up or free space available for data storage and for online and offline data access. What happens if the storage space is not enough to process data? This is when data segments come in handy to keep track of storage details and enhance data storage (R, 2015).

Creating a Tablespace:

To create a tablespace, you can use the CREATE TABLESPACE statement. Here's a basic example:

```
CREATE TABLESPACE your_tablespace  
DATAFILE 'your_datafile.dbf' SIZE 100M;
```

This statement creates a tablespace named your_tablespace with a data file named your_datafile.dbf and an initial size of 100 megabytes.

Viewing Tablespaces:

You can query the DBA_TABLESPACES view or USER_TABLESPACES view to see information about existing tablespaces.

```
SELECT tablespace_name, status, contents  
FROM dba_tablespaces;
```

This query shows the names, status, and contents (e.g., Permanent, Temporary) of all tablespaces.

Altering Tablespaces:

You can use the ALTER TABLESPACE statement to modify the attributes of an existing tablespace. For example, to add a data file to a tablespace:

```
ALTER TABLESPACE your_tablespace  
ADD DATAFILE 'your_additional_datafile.dbf' SIZE 50M;
```

Dropping Tablespaces:

To remove a tablespace and its associated data files, you can use the DROP TABLESPACE statement. Be cautious when dropping tablespaces, as it permanently removes all data stored in that tablespace.

```
DROP TABLESPACE your_tablespace INCLUDING CONTENTS AND DATAFILES;
```

Tablespace Quotas:

You can control the amount of space a user or schema can use within a tablespace by setting quotas. To grant a user a quota on a tablespace:

```
ALTER USER your_user QUOTA UNLIMITED ON your_tablespace;
```

or with a specific limit:

```
ALTER USER your_user QUOTA 50M ON your_tablespace;
```

Moving Tables and Indexes between Tablespaces:

To move a table or index from one tablespace to another, you can use the ALTER TABLE or ALTER INDEX statement with the MOVE TABLE or MOVE INDEX clause.

```
ALTER TABLE your_table MOVE TABLESPACE new_tablespace;
```

These are basic operations related to working with tablespaces in Oracle. Keep in mind that proper planning and understanding of your database's storage requirements are essential for effective tablespace management.

2.4.3 Managing Users and Permissions

While you are performing several database administration tasks such as creating a database and managing memory, one of the tasks that a administrator performs routinely is to manage the users of the database and assign appropriate permission levels to the users. In this topic, you will manage users and permission levels (R, 2015)

Database access for a user is primarily dictated by the privileges granted to that user. There might be times when you will want to provide a user with one permission privilege and restrict it for another user. Oracle provides features that equip you to vary privileges based on user needs, increasing the control you have over your databases (R, 2015).

Predefined User Accounts

The Oracle database provides you with a set of user accounts that guarantee maximum data- base security by using user names, and passwords. These accounts allow you to administer privileges such as CREATE ANY TABLE, or ALTER SESSION. Oracle also provides a few predefined user accounts.

Predefined User Accounts Type	Purpose
Administrative accounts	These accounts are responsible for performing the administration tasks. They are SYS, SYSTEM, SYSMAN, and DBSNMP. The SYS and SYSTEM accounts are responsible for the security privileges, The SYSMAN account is used to perform Oracle Enterprise Manager administration tasks, and the DBSNMP account is used to monitor and manage the database.
Sample schema accounts	These accounts provide access to examples in Oracle Database documentation and instructional materials. They are unlocked and their passwords must be reset before using them
Internal accounts	These accounts enable Oracle Database features or components to have their own schema.

2.4.4 Administrative User Accounts

Administrative accounts in Oracle enable you to perform database administration tasks and all databases use these accounts. They include:

- SYS,
- SYSTEM,
- SYSMAN, and
- DBSNMP,

where SYS and SYSTEM are automatically created once Oracle is installed. The DBA enables users to access the database by granting them various permissions through specific user accounts, which are identified using a valid user name. A user account is identified using the user name, where authentication is done using a password and checking is done to verify if the account is locked earlier. If locked, the DBA unlocks and resets these accounts (R, 2015).

Oracle provides some default administrative user accounts.

Administrative User Account	Functions
CTXSYS	The Oracle Text account is a schema used for searching online help in Oracle Application Express.
DBSNMP	Used to monitor and manage databases.
MDDATA	Used to store router data.
DMSYS	Used to perform data mining operations
OUTLN	Used to manage metadata associated with stored outlines.
SI_INFORMTN_SCHEMA	Used to store information about views for SQL and Still Image Standard
SYS	Used to perform database administration tasks

2.5 User Privileges

Users' privileges are specific rights the DBA assigns to control data access and maintain data- base security. These privileges are provided to them during user account creation to enable the user to connect to the database, update data, and run queries (R, 2015).

Users are basically provided with two types of privileges: system privileges and object privileges.

Privileges	Enables a User To
System privileges	Perform an action on any schema object. For example, the privilege ALTER TABLE allows a user to alter tables in the schema associated with a particular user.
Object privileges	Perform a particular action on a specific schema object. For example, using object privileges, a user can select a particular row from a specific table

Roles

A role is a group of privileges commonly granted to multiple users at the same time. Roles can be created or dropped, and privileges can be granted or revoked from them by the DBA (R, 2015). A role can be created by using the CREATE ROLE statement. The syntax is: CREATE ROLE role_name;

A few predefined roles are available in the database by default each of which is used for a unique purpose.

Role	Purpose
CONNECT	Contains the necessary privileges for the user to connect to the database, and perform basic actions. It includes only CREATE SESSION privilege
RESOURCE	Contains an expanded list of CREATE privileges for more powerful users, like application developers who usually create objects required for development
DBA/SYSDBA	Contains all system level privileges in the database WITH ADMIN OPTION. SYSDBA is the most powerful role in the database with the startup, and shutdown privileges
EXP_FULL_DATABASE	Contains sufficient privileges to export the entire database using the Oracle's Export utility. Most privileges in this role center

	around providing enough SELECT privileges on all tables to enable read permission for all data.
IMP_FULL_DATABASE	Contains sufficient privileges to import the entire database using the Oracle's Import utility. Most privileges in this role center around providing enough CREATE and INSERT privileges to recreate all objects during an import.
SELECT_CATALOG_ROLE	Contains the SELECT privileges on all data dictionary objects. This role is frequently used to allow users to view the data dictionary without the need to give them the DBA role.

Password

A password is an arbitrary alphanumeric string that a user inputs in order to log on to the data- base. Password authentication provides the DBA with a way to maintain, and manage the security of the database by allowing only authenticated users to access it. Every password in Oracle follows a set of rules for its creation, existence, and expiry, as described in the password policy. Further, Oracle protects these passwords by storing them as password hashes, which are further stored in the SYS schema, such that none other than the DBA has access to the schema (R, 2015).

The DBA also determines the complexity of the password by imposing some limitations to its creation and existence, such as:

- Automatic expiry of passwords after the specified days of their creation.
- Automatic lockout of the system after a set number of login failures.
- Presence of a minimum number of alphanumeric and special characters.
- Minimum length of the password.

Oracle Administration Assistant for Windows

Oracle provides you with a graphical tool called Oracle Administration Assistant for Windows, for easy configuration of administrator, user, and other roles. It enables you to:

- Create both local and external Operating System database roles for Windows users.
- Grant local and external Operating System database roles to users or groups.
- Configure a DBA and DBO to access the database without a password.
- Configure automatic startup and shutdown of the database with Oracle services
- Create multiple instances.
- Modify Oracle home Windows registry parameters without using regedit to change the registry.
- Start and stop oracle services

The GRANT Statement

Using a GRANT statement, users can be provided with privileges either at the system level or at the object level to make changes to the database. System-level privileges often allow a user to make global changes that can affect the entire database (R, 2015).

The syntax for implementing system-level privileges is:

GRANT system_privilege[,...] TO [user | role | PUBLIC] [WITH ADMIN OPTION];

where:

- GRANT: Assigns privileges.
- TO [user | role | PUBLIC]: Determines the users who avail the privileges.
- WITH ADMIN OPTION: Specifies that the grantee be allowed to assign the specified privileges. If this clause is omitted, then the ADMIN OPTION will not be given to the grantee.

Similarly, object privileges specify the actions users can take on specific objects. For granting object privileges, you should either own the specific object, possess the GRANT ANY OBJECT PRIVILEGE system privilege, or be assigned with the WITH GRANT OPTION clause previously.

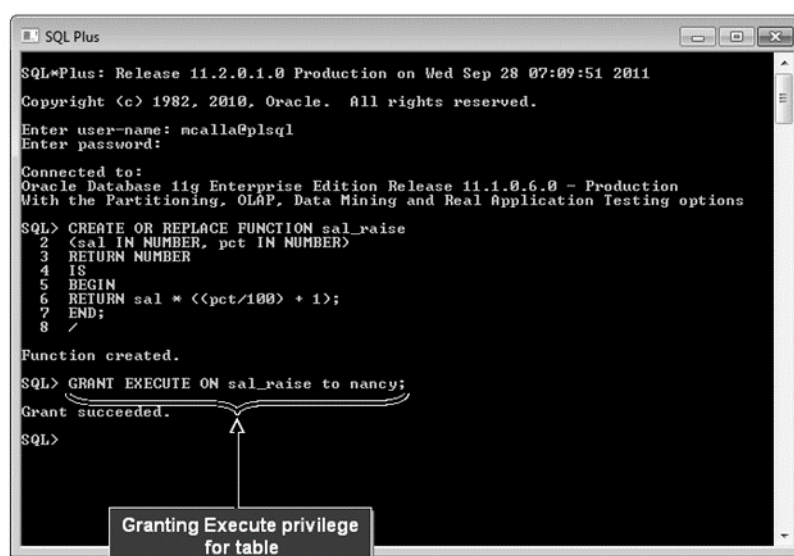
This can be granted using the syntax:

***GRANT privilege ON object
TO [user | role | PUBLIC] [WITH GRANT OPTION];***

where:

- GRANT: Assigns privileges to objects.
- WITH GRANT OPTION: Allows the grantee to pass the object privilege on to other users.

You can also grant privileges to functions and procedures using the codes: GRANT EXECUTE ON object to user; The execute privilege will enable the execution and compilation of the functions and procedures.



```

SQL*Plus: Release 11.2.0.1.0 Production on Wed Sep 28 07:09:51 2011
Copyright (c) 1982, 2010, Oracle. All rights reserved.
Enter user-name: mcalla@plsql
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> CREATE OR REPLACE FUNCTION sal_raise
2 (&sal IN NUMBER, pct IN NUMBER)
3 RETURN NUMBER
4 IS
5 BEGIN
6 RETURN sal * (<pct/100> + 1);
7 END;
8 /

Function created.
SQL> GRANT EXECUTE ON sal_raise to nancy;
Grant succeeded.
SQL>
  
```

Granting Execute privilege for table

Some of the other privileges are listed in the table below.

Privilege	Allowable action
ALTER DATABASE	Alter the configuration of the physical database
CREATE ALTER ANY TABLE	Create or alter tables to be owned by another user.
DROP ANY TABLE	Drop tables owned by any user.
SELECT ANY TABLE	Select from tables owned by any user.
INSERT ANY TABLE	Insert new data into tables owned by any user.
UPDATE ANY TABLE	Modify existing data in tables owned by any user
REFERENCE	Create a constraint that will refer to the data in the table.
INDEX	Create an index on tables using the create index statement.
DELETE	Delete rows from a table using the delete statement.
ALL	Assign all privileges to a user at one stretch.

2.5.1 How to Create Users and Assign Privileges in Oracle

You must first be in the System/SYSDBA connection in order to create users and assign them privileges.

Create a User

Use the CREATE USER statement to create a new user.

CREATE USER your_username IDENTIFIED BY your_password;

NOTE: *In newer versions of Oracle, you will need to add a c## before the user's name for example:*

CREATE USER C##TestUser IDENTIFIED BY password1;

Granting Roles and Privileges

You can grant roles and privileges to the new user. For example, to grant the user the CONNECT and RESOURCE roles.

GRANT CONNECT, RESOURCE TO your_username;

The CONNECT role provides basic connectivity, and the RESOURCE role provides the user with the ability to create tables, indexes, and other schema objects.

2.6 Assigning Specific Privileges

System Privileges

Grant specific system privileges using the GRANT statement.

GRANT CREATE SESSION, CREATE TABLE TO your_username;

This grants the user the ability to create sessions and tables.

Object Privileges:

Grant specific object privileges on a table or other schema objects.

GRANT SELECT, INSERT, UPDATE, DELETE ON your_table TO your_username;

Granting Privileges with Options:

You can use the WITH GRANT OPTION clause to allow the user to further grant the privileges to other users.

GRANT SELECT ON your_table TO your_username WITH GRANT OPTION;

2.6.1 The REVOKE Statement

The privileges granted upon a user may not be required after some time. By using the REVOKE statement, you can withdraw the system level privileges granted to the user using the code:

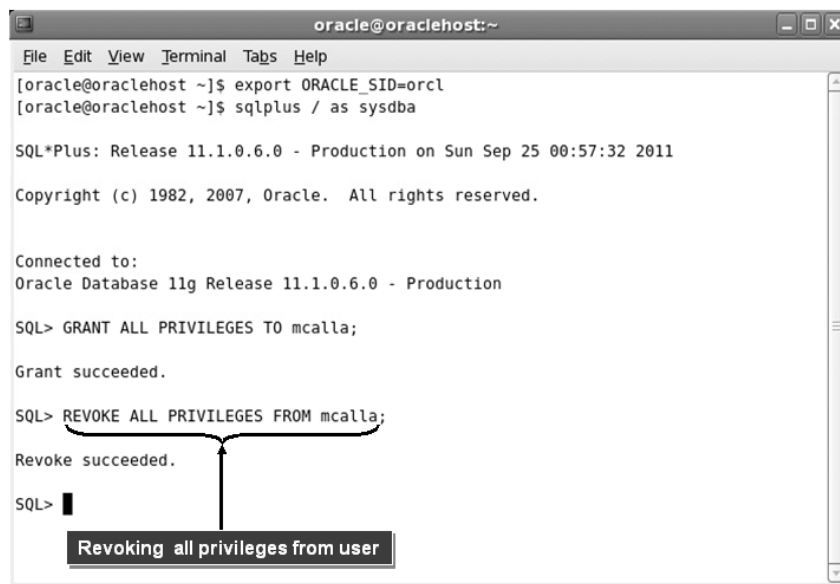
REVOKE privilege[,...] FROM user[,...];

For revoking all allowed privileges, you can use the

REVOKE ALL PRIVILEGES FROM user; statement.

You can revoke access rights for an object using the REVOKE SELECT on table_name FROM user; statement.

Further, you can revoke privileges from procedures, or functions using the revoke execute on object from user; statement (R, 2015).



```
oracle@oraclehost:~  
File Edit View Terminal Tabs Help  
[oracle@oraclehost ~]$ export ORACLE_SID=orcl  
[oracle@oraclehost ~]$ sqlplus / as sysdba  
  
SQL*Plus: Release 11.1.0.6.0 - Production on Sun Sep 25 00:57:32 2011  
  
Copyright (c) 1982, 2007, Oracle. All rights reserved.  
  
Connected to:  
Oracle Database 11g Release 11.1.0.6.0 - Production  
  
SQL> GRANT ALL PRIVILEGES TO mcalla;  
  
Grant succeeded.  
  
SQL> REVOKE ALL PRIVILEGES FROM mcalla;  
  
Revoke succeeded.  
  
SQL> █
```

Revoking all privileges from user

Please note that you will always need to add C## every time you refer to your username.

3 Practical Exercises

- Create a user called sam_smith with the password smith12345. Assign select privileges to sam_smith for the TRANSACTIONS table.

Learning Unit 3: Creating Other Database Objects

Learning objectives addressed in this learning unit:

LO1: Create indexes.

LO2: Apply sequences to a table.

LO3: Write a CREATE VIEW statement.

LO4: Create and use synonyms.

LO5: Evaluate the importance of objects in managing the database.

My notes

1 Creating Other Database Objects

You have used the SQL commands to create databases and modify data within tables. Databases can be more efficiently managed when you optimize data storage, enhance data access, enable data uniqueness, and ensure efficient querying. Managing data by using other database objects such as indexes, sequences, views, and synonyms will ensure better data management. In this lesson, you will create indexes, sequences, views, and synonyms to manage databases (R, 2015).

Increasing use of databases entails more efficient ways of managing data. Consider a bank which maintains a database to manage client accounts. Over time, the clientele swells, and the client account tables the bank maintains begin to increase in size; online transactions start taking longer than usual. Your database now needs to be constantly monitored and its storage has to be regularly optimized to obtain specific results. Do you think that this can be achieved using the same methods of managing databases? Oracle provides various database features such as indexes and views to enhance the manageability of your database(R, 2015).

2 Create Indexes

You have worked with various SQL commands. With increasing number of users and multiple users accessing a database all at once, there will be a need to facilitate faster retrieval of data. In this topic, you will create indexes.

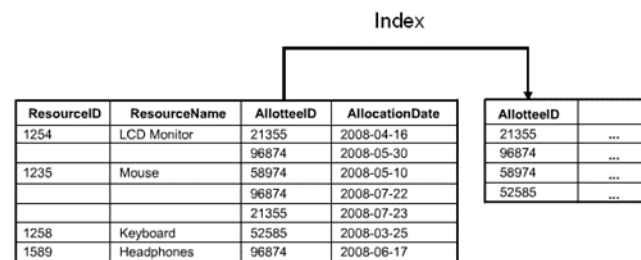
When you buy a book, you generally would look at the index to know what is actually covered in that book. So that in a short time, you are able to get an idea about the book and the coverage of the content. Similarly, if you create an index for your tables in the database, you can retrieve data very quickly, saving time (R, 2015).

Indexes

Definition:

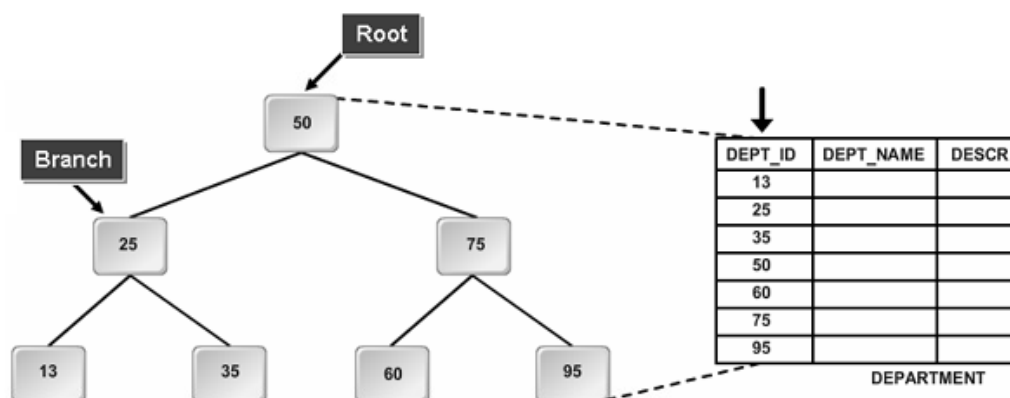
An index is a structure created on a table that provides for easy and quick access to data contained in the table. Updated automatically as new rows are added to or deleted from the table, indexes help drastically improve query performance. You can create indexes on multiple columns of the table that are frequently queried. Also, you can create multiple indexes using different combinations of the columns of the table.

In most relational database applications, indexes are created automatically on the primary or unique key of the table. Indexes are optional structures and you can create or drop indexes without affecting the table or other indexes (R, 2015).



The B*Tree Index

The B*Tree index is the most common type of index in a database. It has a logical structure similar to a tree, except that it is represented upside down. The root is at the top of the tree, and the branches and leaves grow downward. Each entry in the index resides in a node (R, 2015).



The CREATE INDEX statement enables you to create an index on columns in a table. The syntax for creating an index is:

```
CREATE INDEX index_name
ON table_name (column_list...) PCTFREE n
INITRANS n
MAXTRANS n
STORAGE ( INITIAL n NEXT n PCTINCREASE n MINEXTENTS n MAXEXTENTS
n
);
```

where,

- **CREATE INDEX:** Specifies the name of the index to be created.
- **ON clause:** Specifies which table the index will be associated with.
- **column_list:** Specifies the column or columns to create the index for.
- **PCTFREE:** Specifies the percentage of free space that should be reserved for future updates of the values in the index.
- **INITTRANS:** Specifies the initial number of transaction entries per data block.
- **MAXTRANS:** Specifies the maximum number of transaction entries per data block.
- **STORAGE:** Specifies the extent allocation for the index.
- **INITIAL and NEXT:** Specifies the sizes of the first and second extents for the index.
- **PCTINCREASE:** Specifies the percentage for determining the size of the subsequent extents.
- **MINEXTENTS:** Specifies the minimum number of extents that the index may have.
- **MAXEXTENTS:** Specifies the maximum number of extents that the index may have.

Here's an example:

```
CREATE INDEX emp_index
ON employees (employee_id)
PCTFREE 10
INITTRANS 2
MAXTRANS 5
STORAGE (
    INITIAL 64K
    NEXT 64K
    PCTINCREASE 50
    MINEXTENTS 1
    MAXEXTENTS 10
);
```

```
CREATE INDEX emp_index
ON employees (employee_id)
PCTFREE 10
INITTRANS 2
MAXTRANS 5
STORAGE (
    INITIAL 64K
    NEXT 64K
    PCTINCREASE 50
    MINEXTENTS 1
    MAXEXTENTS 10
);
```

In this example, an index named `emp_index` is created on the `employee_id` column of the `employees` table with specified storage and other options. Adjust the values according to your specific requirements (R, 2015).

2.1 Create Sequences

You have created indexes. When managing large volumes of data, you will need to ensure that every row in a table is unique to avoid data redundancy and duplication. In this topic, you will apply sequences to a table (R, 2015).

You are by now able to manage large data using indexes. But how do you maintain data integrity of a database? Using primary keys, you can implement data integrity. Is it really that easy to implement unique primary keys for huge data? Yes, you can implement uniqueness for each table in a database by using sequences (R, 2015).

A sequence is an incremental number generator frequently used to generate unique identifiers for primary key columns in a table. It does not allocate space in a database, but is defined with a starting number and an increment, with a maximum of 28 digits. Databases maintain data integrity by ensuring that every row in each table is unique by using a primary key constraint in conjunction with sequences (R, 2015).

The CREATE SEQUENCE Statement

The CREATE SEQUENCE statement is used to create a sequence to generate unique keys for individual rows of data. The syntax for the CREATE SEQUENCE statement is:

```
CREATE SEQUENCE sequence_name  
[START WITH n ][INCREMENT BY n  
[MINVALUE n | NOMINVALUE] [MAXVALUE n | NOMAXVALUE] [CYCLE |  
NOCYCLE]  
[CACHE n | NOCACHE];
```

In the above syntax,

- **START WITH:** Specifies the number to begin the sequence with.
- **INCREMENT BY:** Specifies the interval between sequence numbers.
- **MINVALUE and MAXVALUE:** Specifies the lowest and highest number allowed for a descending and ascending sequence respectively.
- **NOMINVALUE or NOMAXVALUE:** Specifies that the sequence is independent of any minimum or maximum value, but rather it depends on the resources and processing power of their hardware.
- **CYCLE:** Specifies the clause indicating that the sequence will continue to generate numbers after it has reached its minimum or maximum value. When the ascending clause reaches its maximum value, the sequence will start over from its minimum value.
- **NOCYCLE:** Specifies that once the sequence hits its upper or lower bound, then it becomes unusable. The default is CYCLE.
- **CACHE clause:** Specifies the number of values from the sequence to cache in memory for faster access. The minimum value for this clause is 2. For non-cycling sequences, the maximum value for the CACHE clause is equal to the maximum value of the sequence.
- **NOCACHE clause:** Specifies that no value from the sequence is cached in memory. The default setting for a sequence is CACHE with a value of 20.

Example of a Sequence

The SEQ_EMPLOYEE_ID sequence starts with 1, is incremented by 1, and will have a minimum value of 1. The sequence will have a maximum value of 999999999 and will cycle back to 1 once the maximum value is reached. Also, Oracle will cache 10 values in memory at a time from this sequence.

```
CREATE SEQUENCE seq_employee_id  
START WITH 1  
INCREMENT BY 1  
NOMINVALUE MAXVALUE 999999999  
CYCLE/NO CYCLE  
CACHE 10;
```

How to Apply Sequences

Procedure Reference: Create a Sequence

To create a sequence:

1. Enter CREATE SEQUENCE sequence_name.
2. Enter [START WITH n] to specify the starting value of the sequence.
3. Enter [INCREMENT BY n] to specify the incremental value of the sequence.
4. Enter [MINVALUE n | NO MINVALUE] to specify minimum values for a sequence.
5. Enter [MAXVALUE n | NOMAXVALUE] to specify maximum values for the sequence.
6. Enter [CYCLE | NOCYCLE] to set the cycle.
7. Enter [CACHE n | NOCACHE]; to specify the cache size.
8. Run the query.

To auto-generate primary keys in Oracle, you can use a sequence along with the CREATE TABLE statement. Here's an example of how you can create a table with an auto-incrementing primary key using a sequence:

```
CREATE SEQUENCE your_table_seq  
START WITH 1  
INCREMENT BY 1  
NOMAXVALUE  
NOCYCLE;  
  
CREATE TABLE your_table (  
id NUMBER DEFAULT your_table_seq.NEXTVAL PRIMARY KEY,  
-- other columns in your table  
column1 datatype1,  
column2 datatype2,  
...  
);
```

Let's break down the important parts of the code:

CREATE SEQUENCE your_table_seq: This creates a sequence named your_table_seq. You can replace your_table_seq with a name of your choice. This sequence will be used to generate values for your primary key.

START WITH 1: Specifies the starting value of the sequence.

INCREMENT BY 1: Specifies the increment value for the sequence.

NOMAXVALUE: Specifies that there is no maximum value for the sequence.

NOCYCLE: Specifies that the sequence should not cycle when it reaches the maximum or minimum value.

CREATE TABLE your_table: This creates the table named your_table.

id NUMBER DEFAULT your_table_seq.NEXTVAL PRIMARY KEY: This column definition specifies that the id column is of type NUMBER and its default value is set to the next value of the sequence your_table_seq. This effectively auto-generates the primary key for each new row inserted into the table.

You can adjust the data types and add other columns to the your_table as needed.

Here's an example of inserting data into the table:

```
INSERT INTO your_table (column1, column2, ...)
VALUES ('value1', 'value2', ...);
```

Each time you insert a new row into the table, the id column will be automatically populated with the next value from the sequence.

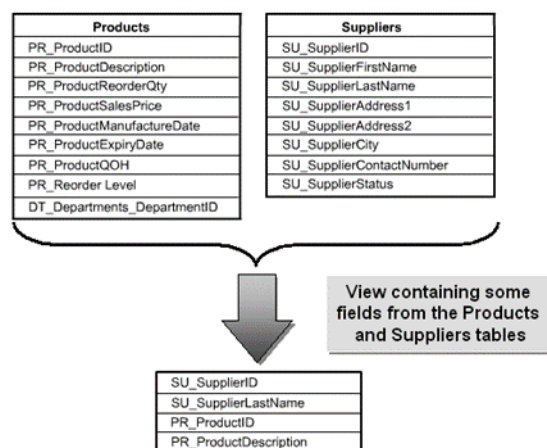
2.2 Create Views

You have applied sequences to a table. To generate specific reports, you often need to refer to some fields and not to all fields in a table. In this topic, you will create various views to generate results that are based on specific user requirements. Have you ever thought why security cameras are being used in an office or a shop? They primarily enable security personnel to view specific areas in multiple sections of the business premises. Similarly, database professionals will access and view only specific information from a table. Views enable you display only specific records that you require from a huge data set (R, 2015).

Definition:

A view can be used to partition the data in a table both vertically and horizontally. A view can be defined to contain only a subset of columns from a base table; this is called vertical partitioning. When the view contains only a subset of rows from a base table, as defined by the WHERE clause of the view, it is called horizontal partitioning.

This type of data separation is very useful for environments where a handful of users need to see only the data that pertains to them. Since views behave exactly like tables, you can also create views based on other views. This technique is called view stacking and can be very useful when you want to simplify a complex query that will generate a simple result set of data (R, 2015).



The CREATE VIEW Statement

The CREATE VIEW statement is used to create a unique view based on table columns. The syntax for creating a view is:

**CREATE [OR REPLACE] [FORCE] VIEW view_name
(column_alias[,...]) AS any valid query;**

In the above syntax,

- **CREATE**: Creates the view.
- **FORCE**: Allows the view to be created even if one or more of the base tables do not exist.
- **column_alias**: Specifies the column aliases that will be used to create the view. It is optional if all the columns in the query are returned with valid column names.

In Oracle, a view is a virtual table based on the result of a **SELECT** query. You can create a view using the **CREATE VIEW** statement. Here's the basic syntax for creating a view:

```
CREATE OR REPLACE VIEW view_name AS  
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

Here's a simple example:

Let's assume you have a table named **employees** with columns **employee_id**, **first_name**, **last_name**, and **salary**. You want to create a view that displays only the **employee_id** and **full_name** (concatenation of **first_name** and **last_name**) for employees with a salary greater than 50000.

```
CREATE OR REPLACE VIEW high_salary_employees AS  
SELECT employee_id, first_name || ' ' || last_name AS full_name  
FROM employees  
WHERE salary > 50000;
```

In this example:

high_salary_employees is the name of the view.

The view includes the **employee_id** and a concatenated column **full_name**.

The data is selected from the **employees** table.

The condition **WHERE salary > 50000** filters the rows based on the salary.

Now, you can query the view just like you would query a table:

```
SELECT * FROM high_salary_employees;
```

2.3 Create Synonyms

You displayed only the required information from the database using views. This simplified the data access and retrieval. To further ease out data manipulation, when handling several tables or users with complicated names, you will want to create short easy referenced names. In this topic, you will use synonyms (R, 2015).

Won't it be always easy and more personal to call your friend by his nickname rather than his lengthy original name? Similarly, you can now create short names or aliases for tables and users so that you can use them easily in queries.

A synonym is a stored alias for an object such as a table or a view in a database. It can be used to shorten long and cumbersome object names. Because there is no data directly related to a synonym, it can be stored directly in the data dictionary along with the name of its corresponding object. It can also be declared as private or public depending on the type of access the administrator wants to provide users with. The types of objects for which the synonym is created are tables, views, packages, procedures, functions, and sequences (R, 2015).

You can create a synonym using the CREATE SYNONYM statement. Here's the basic syntax:

***CREATE [PUBLIC] SYNONYM synonym_name
FOR object_name;***

synonym_name: This is the name of the synonym you want to create.

object_name: This is the name of the object (table, view, sequence, etc.) for which you are creating the synonym.

The PUBLIC keyword is optional and indicates that the synonym is a public synonym, meaning that it is accessible to all users in the database. If you omit PUBLIC, the synonym is private and is only accessible to the user who created it.

Here's an example:

- Creating a private synonym

***CREATE SYNONYM emp_synonym
FOR hr.employees;***

-- Creating a public synonym

***CREATE PUBLIC SYNONYM emp_synonym_public
FOR hr.employees;***

In this example, two synonyms are created:

emp_synonym: This is a private synonym for the hr.employees table.

emp_synonym_public: This is a public synonym for the hr.employees table.

After creating a synonym, you can use it in SQL statements just like you would use the actual object name. For example:

-- Using the private synonym

SELECT * FROM emp_synonym;

-- Using the public synonym

SELECT * FROM emp_synonym_public;

These statements are equivalent to querying the hr.employees table directly. The synonyms provide a convenient and abstract way to reference database objects, making it easier to manage changes to the underlying schema or to simplify SQL statements in your applications (R, 2015).

3 Practical Exercises

- Create a sequence called sales_id that will start at id 560 and increment by one (1).
- Create a view called Transaction_Details that will display the customer ID, broker ID, transaction type and transaction date. In your solution only display the transactions that occurred between the 22 March 2019 and the 25 March 2019. Sample output below:

Sample Results

CUSTOMER_ID	BROKER_ID	TRANS_TYPE	TRANS_DATE
1003	113	Kruger Rands Purchased	22-MAR-19
1002	111	Mortgage Deposit	25-MAR-19

- Create a Synonym for the BROKER table called BRO
- Create an Index on the CUSTOMER TABLE for the FIRST_NAME column

Learning Unit 4: Getting Started with PL/SQL programming

Learning objectives addressed in this learning unit:

- LO1: Compare the features of PL/SQL.
 LO2: Compare the components of the PL/SQL environment.
 LO3: Distinguish between the types of PL/SQL blocks.
 LO4: Explain how to output messages using PL/SQL.
 LO5: Construct PL/SQL Blocks.
 LO6: Design PL/SQL Queries that use Variables.
 LO7: Design PL/SQL Queries with Constants.
 LO8: Construct a PL/SQL query with a Cursor.
 LO9: Explain how to manage data using cursors.
 LO10: Motivate how and why to use:
- The implicit cursor attributes;
 - The explicit cursor attributes.
- LO11: Explain how to use the following clauses:
- o FOR UPDATE;
 - o CURRENT OF.
- LO12: Create PL/SQL queries that use cursor variables.
 LO13: Explain how to retrieve data using the cursor with looping mechanisms.

My notes

1 Introduction

You have used SQL commands to create databases and modify data within tables. Although SQL allows you to directly write queries and manipulate data, it cannot be used to execute procedural codes with conditional, iterative, and sequential statements. PL/SQL, which is Oracle's procedural language extension to SQL, enables you to overcome this limitation. In this lesson, you will use the PL/SQL commands to work with a database efficiently (R, 2015).

When you are managing large amounts of data, you may need to process a set of queries simultaneously. Sometimes, you may need to process information or data based on the processing result of an earlier query or command. It will be tedious to execute individual queries using SQL commands. PL/SQL is Oracle's procedural programming language extension to SQL, and it allows incorporation of complex logic and business rules into database data management (R, 2015).

2 Use PL/SQL Blocks

In this lesson, you will use PL/SQL commands to manage data. You have used SQL commands to work with a database. However, with SQL, you cannot execute several conditional queries successively. In this topic, you will use PL/SQL blocks to work with data.

Procedural Language/Structured Query Language (PL/SQL) is a procedural language developed by Oracle as an extension to SQL. PL/SQL combines the capabilities of procedural logic with the simplicity of basic SQL commands. It is modular and portable and enables execution of procedural techniques and logic on a database with conditional, iterative, and sequential statements. These programs usually run on the database server. However, in some Oracle products, the PL/SQL engine resides on a client. It is commonly used to write data-centric programs that enable data manipulation in an Oracle database (R, 2015).

2.1 Features of PL/SQL

PL/SQL provides many features that allow you to write effective code to manipulate your data- base. As it is a procedural language extension of SQL, it allows you to use SQL commands, functions and operators. It supports SQL data types and allows SQL statements to be constructed and executed. PL/SQL is a block-structured language, meaning that blocks of code are passed to the Oracle server, instead of one SQL command statement being passed at a time, thus increasing performance and decreasing network traffic. A PL/SQL program is portable in any platform on which Oracle works (R, 2015).

Advantage	Description
Block structure	PL/SQL is a language composed of blocks of code that can be nested within each other. Each block refers to a particular task or module, allowing you to create modular code. Procedures, functions, and anonymous blocks that constitute a PL/SQL program are logical blocks, which can be nested inside one another.
Procedural Language Capabilities	PL/SQL defines procedural language constructs such as conditional logic and loops. Conditional logic includes IF..THEN..ELSE statements, and loops include WHILE statements, which are used to execute a set of statements repeatedly until a specific condition is met.
Error Handling	Certain types of errors, called exceptions, can be trapped during PL/SQL program execution. Once an exception is trapped, corrective or diagnostic actions can be performed.
Complete Portability	You can run applications written in PL/SQL on any operating system and platform where the Oracle database runs.

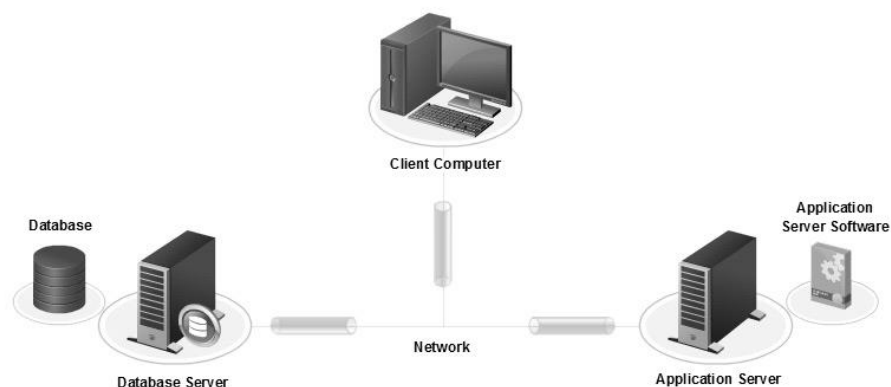
Tight Integration with SQL	PL/SQL allows you to use SQL commands, functions and operators. You do not have to translate between SQL and PL/SQL data types. You can also construct SQL statements dynamically in PL/SQL.
----------------------------	--

2.2 Components of the PL/SQL Environment

A typical PL/SQL environment includes several major components.

PL/SQL Environment Component	Description
Database Server	A computer which runs the Oracle database. PL/SQL runs on the database server. A Database Management System, an Oracle software unit, is installed on the database server.
Database	A set of tables and data that constitute the information stored.
Application Server	A computer where you store and execute your application codes.
Application Server Software	A special software program used to run and manage applications on the application server.
Client Computer	A computer where the client and server programs are executed.
Network	A channel through which the application server, client computers and database server communicate with each other

The above figure shows the PL/SQL environment.



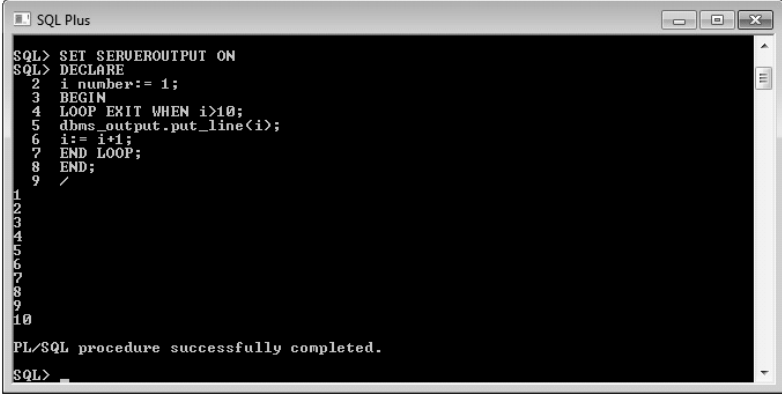
The PL/SQL Engine

The PL/SQL engine compiles and executes PL/SQL blocks. It can be installed on an Oracle server or in an application development tool such as Oracle forms or reports. Either way, the PL/SQL engine accepts and executes any valid PL/SQL block or

subprogram. It is a separator, splitting the SQL and PL/SQL statements. A PL/SQL block is a combination of SQL and procedural statements that are submitted as one request. The PL/SQL engine executes procedural statements but sends SQL statements to the SQL engine in the Oracle database. The PL/SQL code that you write is processed either by the client-side or server-side PL/SQL engine depending on where you store your code (R, 2015).

SQL*PLUS

SQL*Plus is an Oracle-developed application with a command line interface that you can use to submit SQL statements and PL/SQL blocks for execution. The results are displayed in an application or command window. It provides developers and end users with an interface to either connect to the Oracle server or disconnect from the server. It is independent of any database versions or operating systems, allowing a single workstation to seamlessly connect to all available Oracle database servers. SQL commands are stored in special files called scripts, which can be saved, edited, and run from the prompt (R, 2015).



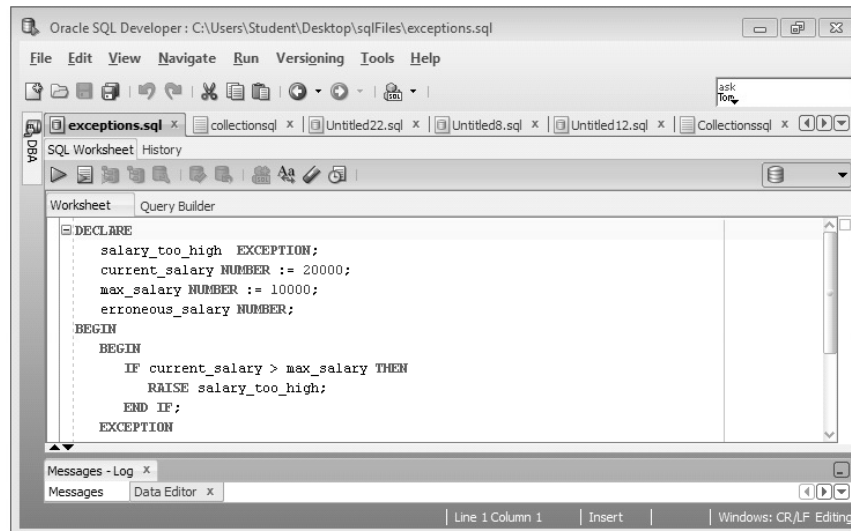
```
SQL Plus
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
2  i number:= 1;
3  BEGIN
4  LOOP EXIT WHEN i>10;
5  dbms_output.put_line(i);
6  i:= i+1;
7  END LOOP;
8  END;
9  /
1
2
3
4
5
6
7
8
9
10
PL/SQL procedure successfully completed.
SQL>
```

Advantages of SQL*Plus

SQL*Plus has many features that make working with the database easy and efficient. It has an editing environment for developing your code, facilities for saving and running script files, methods of using local variables, and powerful formatting commands that can be used to create reports. You can also describe the structure of database objects such as tables. SQL*Plus also maintains a working area in memory, called the SQL*Plus buffer, that contains your most recently executed SQL statement or PL/SQL block (R, 2015).

ORACLE SQL Developer

Oracle SQL Developer is a free graphical tool designed to improve your productivity and enable you to perform everyday database tasks efficiently. You can easily create and debug stored procedures, test SQL statements, browse and manage database objects, edit and debug PL/SQL statements and create reports. Oracle SQL Developer has two main navigation tabs: the Connections tab and the Reports tab. Using the Connections tab, you can browse database objects and users to which you have access. Using the Reports tab, you can run predefined reports, or create and add your own reports (R, 2015).



2.3 Write a Simple PL/SQL Program Block

You have identified the basics of PL/SQL. PL/SQL is a structural programming language that enables you to nest blocks of code. In this topic, you will write a simple PL/SQL program block.

A language whether human or computer has a defined structure, syntax, vocabulary, and a character set. Before using any new language, we must first know the rules that govern its creation and usage. PL/SQL is a procedural language that contains meaningful blocks of code.

Using the basic components to create a logical block will enable you to create efficient procedural blocks of code (R, 2015).

2.4 PL/SQL Blocks

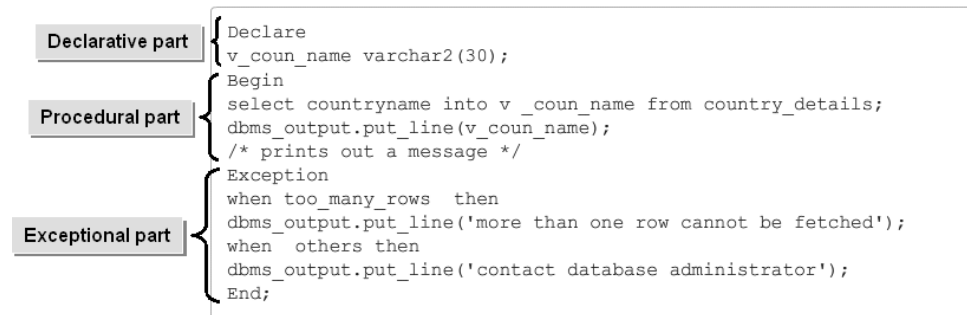
The PL/SQL block is a basic set of SQL and PL/SQL statements that enable you to solve the specific problem. A PL/SQL block consists of three sections: An optional Declaration section, a mandatory Execution section and an optional Exception section.

The Declaration section of a PL/SQL block begins with the reserved word DECLARE and continues to the beginning of the Execution section. The Declaration section is optional and is used for declaring program constructs such as variables (R, 2015).

The Execution section starts with the reserved keyword BEGIN, and ends with either the key- word EXCEPTION, or if that section is not included, the keyword END. The Execution section is mandatory and is where you define the process for any task to be performed. This section contains program code, which is composed of individual procedural and SQL statements (R, 2015).

Every statement in this section must end with a semicolon (;). Comments can be used to document code.

The Exception section follows the Execution section and starts with the reserved word **EXCEPTION**. Exceptions are warnings or error conditions that may be generated by the system or defined by the user. The code in the **EXCEPTION** section can identify the exception and take appropriate action (R, 2015).



2.5 Types of PL/SQL Blocks

PL/SQL lets you create two type of blocks, anonymous and named.

Anonymous: As the name suggests, this type of block has no name, which means that it can- not be called by any other block. It lacks a header section. An anonymous block serves as a container that executes PL/SQL statements, which usually include calls to procedures or functions. It provides scope for identifiers and exception handling within a larger program (R, 2015).

Named: Named blocks are either procedures or functions and can be called by other blocks. Most of the code you write will be in named blocks. A named block is stored in the database and has a header section (R, 2015). The name, list of formal parameters, and any return type of a named PL/SQL block are defined by the block header. The syntax for a named block is

:

```
FUNCTION name [ ( parameter [, parameter ... ] ) ] RETURN return_datatype
IS [ declaration statements ]
BEGIN
executable statements
EXCEPTION
[exception handler statements ]
END [ name ];
```

2.6 Output Messages in PL/SQL

PL/SQL is designed to work directly with the database to manipulate data. Oracle has provided some built-in functionality that will help you generate output on the screen.

The first step is to set the server output to **ON**. Because the default is **OFF**, you must instruct SQL*Plus to display output on-screen every time you start a new session. You can set the server output using **SET SERVEROUTPUT ON**. If you do not, the PL/SQL block may execute successfully, but you will not see the result.

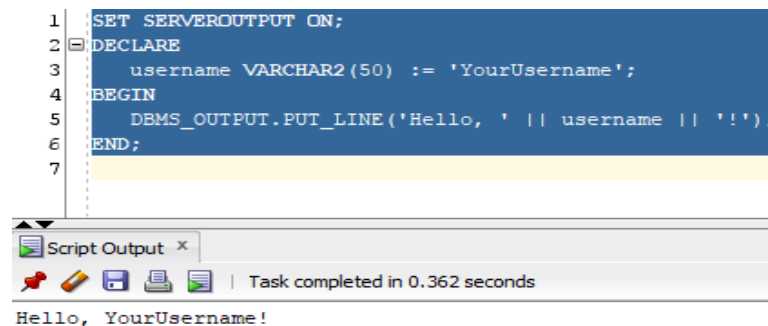
Next, enter the BEGIN keyword to inform the PL/SQL engine about the beginning of the PL/SQL block's mandatory Execution section.

The next step is to display a message; you can enter:

dbms_output.put_line('your message'); in the EXECUTION section after the BEGIN STATEMENT.

Below is code that shows how you do this:

```
SET SERVEROUTPUT ON;  
DECLARE  
    username VARCHAR2(50) := 'YourUsername';  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('Hello, ' || username || '');  
END;
```



PL/SQL procedure successfully completed.

PL/SQL

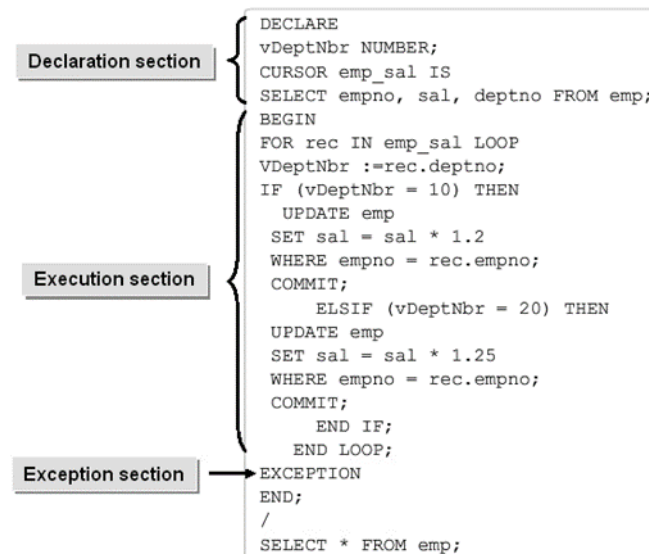
Procedural Language/Structured Query Language (PL/SQL) is a procedural language developed by Oracle as an extension to SQL. PL/SQL combines the capabilities of procedural logic with the simplicity of basic SQL commands. It is modular and portable and enables execution of procedural techniques and logic on a database with conditional, iterative, and sequential statements. The programs written with this language have block structures containing procedures, functions, and blocks as their basic units. These programs usually run on the database server; however, in some Oracle products, the PL/SQL engine resides on the client. It is commonly used to write data-centric programs that enable data manipulation in an Oracle database (R, 2015).

The PL/SQL Block

PL/SQL programs are known as blocks. PL/SQL blocks are broken into three sections: declaration, execution, and exception handling. Among these, the execution section is the only required section in a block. Code within a PL/SQL block is case insensitive, with the exception of literal values that are enclosed in single quotes. PL/SQL provides two types of blocks: named blocks and anonymous blocks. Named blocks are stored in the data dictionary of the database and executed only when called. These blocks can be stored as procedures, functions, or triggers. Anonymous blocks are blocks of

PL/SQL code that are not named, and are executed at any time from any interface that can access the database. While the code for a named block is stored in the database, the code for an anonymous block must be passed to the database at runtime. You can type an entire anonymous block at the SQL*Plus prompt and execute it immediately. All code for an anonymous block must be passed to the database at the time of execution (R, 2015).

Here are the more detailed information on the three sections of a PL/SQL block.



There are various sections in a PL/SQL Block

Declaration: It is indicated by the keyword `DECLARE`. It includes variables, cursors, or special PL/SQL tables that are needed for the execution section. These structures reside only in memory while the block is executing.

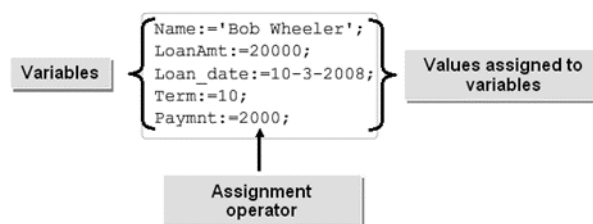
Execution: The `BEGIN` keyword marks the beginning of the execution section. The execution section is the only section of the block that is required, and it drives the entire workload of the block. All procedural logic, which allows a program to make condition-based decisions, is contained within the executable section. This section can contain SQL statements, call other PL/SQL programs, reference any structure created in the declaration section, and can read from and write to data structures within the database.

Exception handling: The exception handling section is indicated by the keyword `EXCEPTION`; it is used to handle any exceptions that are generated by the task done in the executable section and is generally used to describe errors that occur during processing. The exception handling section allows you to define certain program blocks that can be called when any error-prone situations arise during the processing of the execution section. The keyword `END` marks the end of the block (R, 2015).

2.7 Variables

A variable is a temporary memory location that holds a variety of data such that it can be changed at any point during the execution of a block. It is replaced by real data during program execution, but once the execution is complete, the reserved memory is released. Like all storage structures in Oracle, variables must conform to one of Oracle's data types. The length of the variable name is limited to 30 characters. The assignment operator, which is a colon followed by an equal sign (:=), is used to assign a value to a variable (R, 2015).

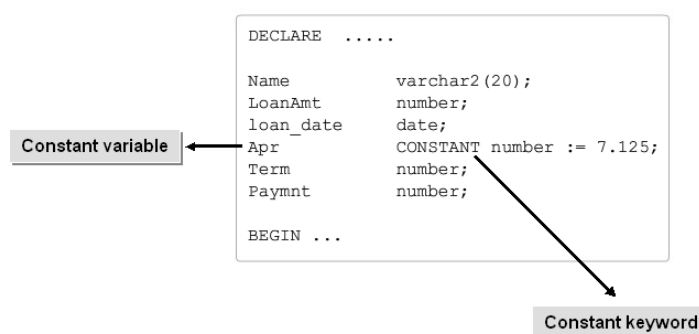
Example:



2.8 Constants

A constant is a temporary memory location that holds data with a specific data type such that the value remains unchanged throughout program execution. Like all storage structures in Oracle, constants also conform to one of Oracle's data types. The length of a constant name is limited to 30 characters. In addition, to declare a variable as a constant, data types are preceded by the keyword CONSTANT (R, 2015).

Example:



Here's an example that demonstrates the use of constants:

```

DECLARE
  -- Declare a constant
  PI CONSTANT NUMBER := 3.14159;

  -- Declare variables
  radius NUMBER := 5;
  
```

```
circumference NUMBER;
```

```
BEGIN
```

```
-- Use the constant in calculations
```

```
circumference := 2 * PI * radius;
```

```
-- Display the result
```

```
DBMS_OUTPUT.PUT_LINE('Radius: ' || radius);
```

```
DBMS_OUTPUT.PUT_LINE('Circumference: ' || circumference);
```

```
END;
```

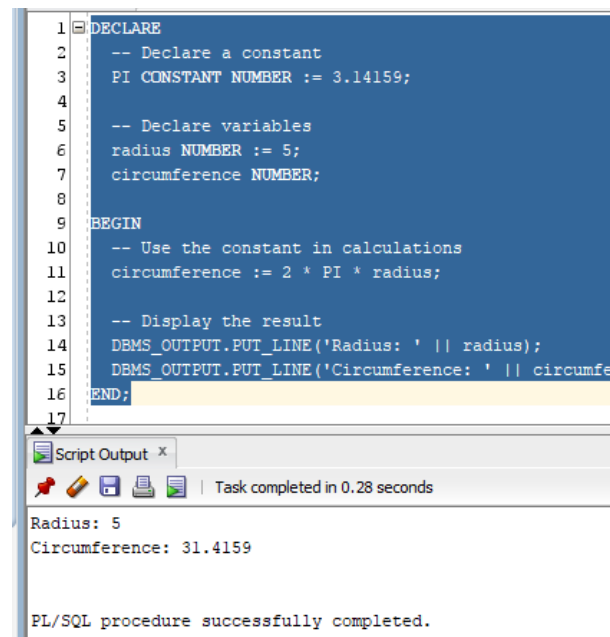
In this example:

PI is declared as a constant with the value 3.14159. Constants are defined using the CONSTANT keyword after the datatype.

radius is declared as a variable and initialized with the value 5.

circumference is declared as a variable but not initialized initially. It is later assigned the value of $2 * PI * radius$.

The DBMS_OUTPUT.PUT_LINE statements are used to display the values of the variables and the calculated circumference.



The screenshot shows a PL/SQL script editor with the following code:

```
1 DECLARE
2   -- Declare a constant
3   PI CONSTANT NUMBER := 3.14159;
4
5   -- Declare variables
6   radius NUMBER := 5;
7   circumference NUMBER;
8
9 BEGIN
10  -- Use the constant in calculations
11  circumference := 2 * PI * radius;
12
13  -- Display the result
14  DBMS_OUTPUT.PUT_LINE('Radius: ' || radius);
15  DBMS_OUTPUT.PUT_LINE('Circumference: ' || circumfe
16 END;
```

Below the editor is a 'Script Output' window showing the results of the execution:

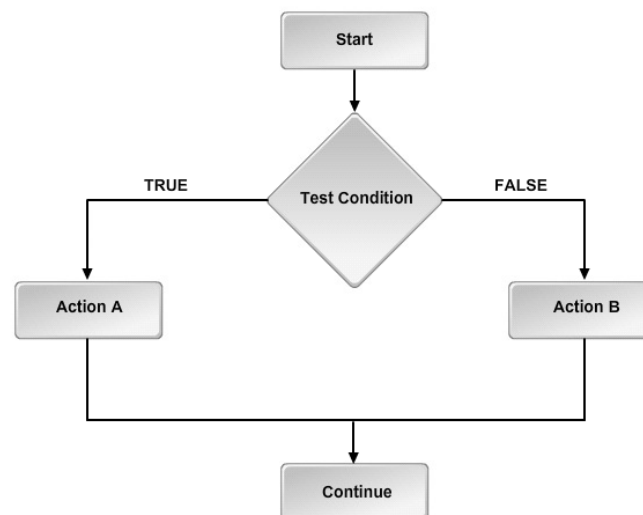
```
Task completed in 0.28 seconds
Radius: 5
Circumference: 31.4159
PL/SQL procedure successfully completed.
```

2.9 Conditional Flow Statements

Conditional flow statements are logical statements used in any procedural language to make decisions based on specified conditions. A condition, which can be either true or false is called the truth value. A control structure will determine the truth value for a given condition, and then perform specific actions based on that truth value. The three types of control structures in PL/SQL are selection, iteration, and sequential (R, 2015).

2.10 Selection Control Structures

Selection control structures are used to select a particular action based on a condition. The selection control structure will determine the truth value of a condition, and then perform one action if the condition is true or another action if the condition is false. Various selection control statements are used to perform a particular action.

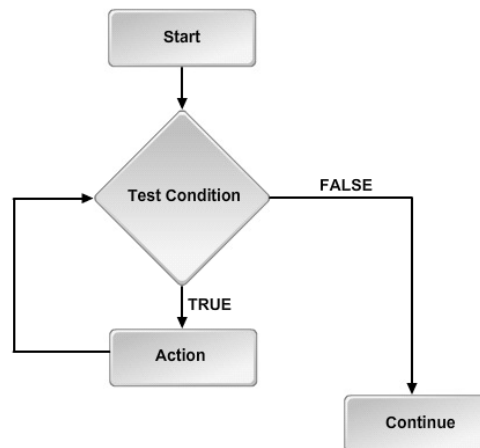


Selection control structures can be implemented with the IF statement, which states that if the IF statement is true, the action is executed, else not. ELSE and ELSEIF clauses are used with the IF statement to provide an alternative action to execute the statements if the condition is false. There is no limit to the number of ELSIF clauses that can be included in an IF statement (R, 2015). The syntax for the IF statement is:

- IF condition THEN
series of actions...
END IF;
- IF condition THEN
series of actions.
ELSE
alternative series of actions.
END IF;

2.11 Iterative Control Structures

Iterative control structures perform a series of actions repeatedly as long as a condition remains true. Once the condition turns to be false, the iteration stops, and the program continues its execution. Iterative control structures enable you to execute a sequence of statements written within the LOOP and END LOOP statements multiple times. However, there can also be an unconditional or forceful exit from the loop by using the EXIT and EXIT WHEN statements (R, 2015).



2.11.1 Iterative Control Statements

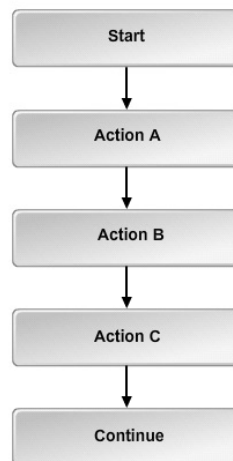
There are three main kinds of iterative control statements used in loops: the basic loop, WHILE loop, and FOR loop. The execution of the loop begins soon after one of the keywords LOOP, WHILE, or FOR is encountered. The WHILE loop is used when the number of iterations is unknown, but the FOR loop is used when the number of iterations is known (R, 2015).

The syntax for the basic loop, WHILE loop, and FOR loop is:

- Basic loop: LOOP
Series of actions...
END LOOP;
- WHILE loop: WHILE condition LOOP
series of actions...
END LOOP;
- FOR loop: FOR counter IN lower_bound..upper_bound LOOP
series of actions...
END LOOP;

2.11.2 Sequential Control Statements

Sequential control structures will execute a series of actions in a sequence, or in a specific order. These commands enable a program to jump from one location in the program to another. Oracle provides the GOTO statement and code labels that are used to redirect the flow of execution from one point in the program to another. The GOTO statement instructs Oracle to stop execution at a certain point in the program and start execution again at another point. The destination indicated by the GOTO statement must be a code label (R, 2015).



2.12 Sequential Control Statements

The GOTO statement uses code labels for making sequential statements. To create a code label, you would enclose the label name within double angle brackets (<< ... >>). The label name must follow the same rules as variables and tables. It must be no longer than 30 characters, must not start with a number or special character, and must not contain any space. Also, the code label must appear immediately preceding an executable statement. For example, a label may appear immediately before an IF statements, but not immediately before an END IF or an END LOOP clause (R, 2015).

DECLARE

vTestNum NUMBER := 10; BEGIN

<<label1>> GOTO label3;

vTestNum := vTestNum * 50;

<<label2>>

vTestNum := vTestNum * 100;

<<label3>>

DBMS_OUTPUT.PUT_LINE('The number is '||vTestNum);

Cursors

END;

Here is an example of using PL/SQL statements with variables.

DECLARE

-- Declare variables of different data types

v_number INTEGER := 42;

v_text VARCHAR2(50) := 'Hello, PL/SQL!';

v_result NUMBER;

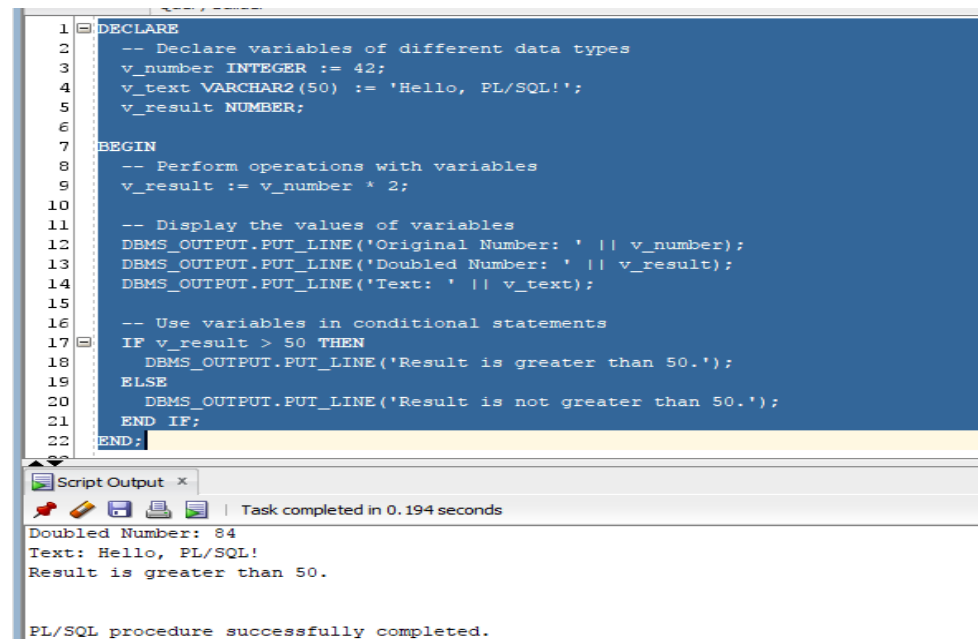
```
BEGIN
-- Perform operations with variables
v_result := v_number * 2;

-- Display the values of variables
DBMS_OUTPUT.PUT_LINE('Original Number: ' || v_number);
DBMS_OUTPUT.PUT_LINE('Doubled Number: ' || v_result);
DBMS_OUTPUT.PUT_LINE('Text: ' || v_text);

-- Use variables in conditional statements
IF v_result > 50 THEN
    DBMS_OUTPUT.PUT_LINE('Result is greater than 50.');
```

```
ELSE
    DBMS_OUTPUT.PUT_LINE('Result is not greater than 50.');
```

```
END IF;
END;
```



The screenshot shows a SQL script execution window with a blue background for the code editor. The code is as follows:

```
1 DECLARE
2   -- Declare variables of different data types
3   v_number INTEGER := 42;
4   v_text VARCHAR2(50) := 'Hello, PL/SQL!';
5   v_result NUMBER;
6
7 BEGIN
8   -- Perform operations with variables
9   v_result := v_number * 2;
10
11  -- Display the values of variables
12  DBMS_OUTPUT.PUT_LINE('Original Number: ' || v_number);
13  DBMS_OUTPUT.PUT_LINE('Doubled Number: ' || v_result);
14  DBMS_OUTPUT.PUT_LINE('Text: ' || v_text);
15
16  -- Use variables in conditional statements
17  IF v_result > 50 THEN
18      DBMS_OUTPUT.PUT_LINE('Result is greater than 50.');
```

```
19  ELSE
20      DBMS_OUTPUT.PUT_LINE('Result is not greater than 50.');
```

```
21  END IF;
22  END;
```

Below the code editor, there is a 'Script Output' window showing the results of the execution:

```
Task completed in 0.194 seconds
Doubled Number: 84
Text: Hello, PL/SQL!
Result is greater than 50.

PL/SQL procedure successfully completed.
```

In this example:

`v_number` is declared as an `INTEGER` and initialized with the value 42.
`v_text` is declared as a `VARCHAR2` and initialized with the string 'Hello, PL/SQL!'.
`v_result` is declared but not initialized initially. It is later assigned the value of `v_number * 2`.
The `DBMS_OUTPUT.PUT_LINE` statements are used to display the values of the variables and the result of the operation.

In PL/SQL, you can work with different types of loops to iterate over a set of statements. The most common types of loops are `FOR`, `WHILE`, and `LOOP`. Here's an example that demonstrates the use of each type:


```
DECLARE
-- Declare variables
v_counter NUMBER := 1;
v_limit NUMBER := 5;

BEGIN
-- FOR Loop
DBMS_OUTPUT.PUT_LINE('Using FOR Loop:');
FOR i IN 1..v_limit LOOP
    DBMS_OUTPUT.PUT_LINE('Iteration ' || i);
END LOOP;

-- WHILE Loop
DBMS_OUTPUT.PUT_LINE('Using WHILE Loop:');
WHILE v_counter <= v_limit LOOP
    DBMS_OUTPUT.PUT_LINE('Iteration ' || v_counter);
    v_counter := v_counter + 1;
END LOOP;

-- LOOP with EXIT WHEN
DBMS_OUTPUT.PUT_LINE('Using LOOP with EXIT WHEN:');
v_counter := 1;
LOOP
    EXIT WHEN v_counter > v_limit;
    DBMS_OUTPUT.PUT_LINE('Iteration ' || v_counter);
    v_counter := v_counter + 1;
END LOOP;
END;
```

Output for the above query

```
Using FOR Loop:
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Using WHILE Loop:
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Using LOOP with EXIT WHEN:
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
```

2.13 Cursors

A cursor is a work area in memory. It is used to access, store and process data in a database. The syntax for a cursor that accepts arguments through a parameter is:
`CURSOR cursor_name (arguments) IS SELECT field_name FROM table_name WHERE condition;`

When a SQL statement is executed, Oracle opens a private work area in system memory called a cursor. Cursors contain information on the SELECT statement and the number of rows of data accessed. This temporary work area is used to store data retrieved from the database into client memory (R, 2015).

A cursor enables you to name the work area and to manipulate the data within. Suppose you want to use a cursor to update the salary of all the salespeople. You can create a cursor with all salary values to complete this task. The set of rows sent to a cursor is called the active set. The cursor automatically determines the number of rows depending on the number of rows in the active set. Even though a cursor stores multiple rows, it can process only one row at a time—called the current row. As you fetch a row, the current row automatically moves to the next row.

There are two types of cursors in PL/SQL: implicit cursors and explicit cursors. Both implicit and explicit cursors have the same functionality, but they differ in the way they can be accessed (R, 2015).

Example of Declaring a Cursor

The syntax for declaring a cursor is:

```
CURSOR cursor_name IS select_statement; For example,  
DECLARE  
CURSOR c1 IS SELECT empno, ename, job FROM emp WHERE deptno = 20;
```

There are two types of cursors.

Implicit Cursor: They are created automatically by Oracle. All SQL statements that handle data directly, including the SELECT, INSERT, UPDATE, and DELETE statements, use implicit cursors. During processing, the cursor is created and opened automatically, and the data is fetched from the database or the user process, and stored in the cursor. The data is processed depending on the SQL statement that is issued (R, 2015).

Explicit Cursor: They are manually created from within PL/SQL. Explicit cursors can process rows in a result set individually, to take a different action for each row based on the values in that row. This provides a flexible way of manipulating data in a database (R, 2015).

They are declared in the declaration section of a PL/SQL block, and are given a name and a definition in the form of a SELECT statement. This provides developers a more

flexible way to manipulate data in large quantities without having to write a section of a program repeatedly to perform a particular task.

2.14 Implicit Cursors

When DML statements are used in a program, Oracle implicitly declares a cursor called an implicit cursor to temporarily store data in system memory. Implicit cursors are created by default either when DML statements such as INSERT, UPDATE, and DELETE statements are executed, or when a SELECT statement that returns just one row is executed (R, 2015).

2.15 Implicit Cursor Attributes

You can use implicit cursor attributes to find the status of a DML operation in a PL/SQL block.

Attribute	Returns
%FOUND	TRUE, if DML statements such as INSERT, UPDATE, and DELETE affect at least one row and also if the SELECT...INTO statement returns at least one row. FALSE, if DML statements like INSERT, UPDATE, and DELETE do not affect any row, and also if the SELECT...INTO statement does not return a row. For example, SQL%FOUND
%NOTFOUND	FALSE, if DML statements like INSERT, UPDATE, and DELETE affect at least one row and also if the SELECT...INTO statement returns at least one row. True if DML statements like INSERT, UPDATE and DELETE do not affect even one row and also if the SELECT...INTO statement does not return a row. For example, SQL%NOTFOUND.
%ROWCOUNT	The number of rows affected by INSERT, UPDATE, and DELETE statements or returned by a SELECT... INTO statement. Here, a 1 is returned, if at least one row is affected or returned and a 0 is returned, if no rows are affected or returned. For example, SQL%ROWCOUNT
%ISOPEN	Always FALSE for implicit cursors because the cursor is closed immediately after the statement is executed. For example, SQL%ISOPEN

Example of Implicit Cursor Attributes

Consider the PL/SQL block that uses implicit cursor attributes as shown:

```
DECLARE var_rows number(5); BEGIN
UPDATE employee SET salary = salary + 1000;
IF SQL%NOTFOUND THEN
  dbms_output.put_line('None of the salaries were updated'); ELSE var_rows :=
  SQL%ROWCOUNT;
  dbms_output.put_line('Salaries for ' || var_rows || 'employees are updated');
END IF;
END;
```

Here, the salaries of all employees in the employee table are updated. The result states: if the salaries of employees are updated, we get the message “Salaries for 1000 employees are updated”, if there are 1000 rows in the employees tables. Or else, we get the message “None of the salaries were updated”.

2.16 Explicit Cursors

Explicit cursors have to be defined in a PL/SQL program when working with SELECT statements that return more than one row of data. A cursor is based on a query and is defined in the declaration section of the PL/SQL block (R, 2015). You can define your own names for explicit cursors, unlike with implicit cursors, whose attributes are referred to as SQL%attribute_name.

Example of an Explicit Cursor

To work with an explicit cursor, you must first declare it. An explicit cursor is defined in the declaration section of a PL/SQL block. Declaring a cursor consists of naming it and specifying a SELECT statement on which it is based.

The syntax is:

```
CURSOR cursor_name IS select_statement; For example,
DECLARE
CURSOR product_cur IS SELECT *
FROM product
WHERE unit_price > 10; where the values of unit_price greater than 10 are placed in the product_cur cursor.
```

Access Explicit Cursors

There are four steps involved when accessing an explicit cursor:

1. First, declare the cursor in the declaration section. Declaring a cursor consists of naming it and specifying a SELECT statement on which it is based.

The syntax is:

```
DECLARE
CURSOR cursor_name IS select_statement;
```

2. The second step is to open the cursor in the execution section. The syntax is:

OPEN cursor_name;

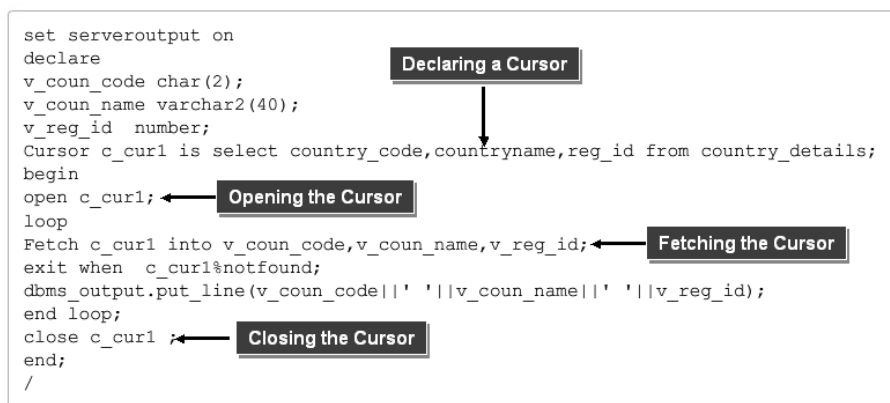
3. The third step is to fetch data from the cursor into PL/SQL variables. The syntax is:

FETCH cursor_name INTO record_name;

4. The last step is to close the cursor in the execution section. The syntax is:

CLOSE cursor_name;

When a cursor is opened, the first row becomes the current row. When data is fetched, it is copied to the record or variables and the logical pointer moves to the next row and it becomes the current row. On every fetch statement, the pointer moves to the next row.



Example of Accessing Explicit Cursors

Consider the code:

```

DECLARE emp_recd employees%rowtype; CURSOR emp_curs IS
SELECT * FROM WHERE salary > 7000; BEGIN
OPEN emp_curs; loop
FETCH emp_curs INTO emp_recd; exit when emp_curs%NOTFOUND;
dbms_output.put_line (emp_recd.first_name || ' ' || emp_recd.last_name
end loop;
CLOSE emp_curs;
END;

```

Here, employees whose salary is greater than 7000 are placed in the cursor emp_curs.

2.17 Explicit Cursor Attributes

Explicit cursor attributes help control data processing when using cursors.

Attribute	Returns
%FOUND	TRUE, if the FETCH statement returns at least one row. FALSE, if the FETCH statement doesn't return a row. For example, CUR/OR_NAME%FOUND
%NOTFOUND	TRUE, if the FETCH statement doesn't return a row. FALSE, if the FETCH statement returns at least one row. For example, CUR/OR_NAME%NOTFOUND
%ROWCOUNT	The number of the row fetched by the FETCH statement. If no row is returned, the PL/SQL block returns an error. For example, CUR/OR_NAME%ROWCOUNT
%ISOPEN	TRUE, if the cursor is already open in the program. FALSE, if the cursor is not opened in the program. For example, CUR/OR_NAME%ISOPEN

2.18 Controlling a Cursor Manually

A cursor is manually controlled using the OPEN, FETCH, and CLOSE statements. An explicit cursor must be manually opened with the OPEN statement before its result set can be processed. Once opened, the FETCH statement advances the current row pointer to the first row in the cursor and loads that row's values into the variables you designate. Once the values are loaded, you can perform any type of processing on the values just like any other variable. When the data in that row is processed, you can issue the FETCH statement again to advance the current row pointer and load the next row. Once all rows in the result set have been processed, you can close the cursor with the CLOSE statement (R, 2015).

The following example shows a sample anonymous block that manually processes a cursor.

```

DECLARE
vItemId NUMBER;
vItemName VARCHAR2;
vRetailPrice NUMBER;
vVendorID NUMBER;
CURSOR cur_item IS
SELECT item_id, item_name, retail_price, vendor_id FROM item
WHERE item_id BETWEEN 5400 and 6000;
BEGIN

```

```
OPEN cur_item;
FETCH cur_item INTO vItemID, vItemName, vRetailPrice, vVendorID;
IF vVendorID = 7 THEN
vRetailPrice := vRetailPrice * 1.1; ELSIF vVendorID = 8 THEN vRetailPrice :=
vRetailPrice * 1.2; END IF;
UPDATE item
SET retail_price = vRetailPrice WHERE item_id = vItemID;
CLOSE cur_item;
END;
```

In this example, a cursor was declared to identify each row in the ITEM table that had an ITEM_ID between 5400 and 6000. In the execution section, the cursor was opened with the OPEN statement. When the FETCH statement was issued, the current row pointer was advanced to the first row in the result set and the values were loaded into the specified variables. That row was processed by the IF statement to update the retail price of the item depending on the vendor who makes the item. The new values were then passed back to the ITEM table with the UPDATE statement. The CLOSE statement was issued to close the cursor. Notice that in this block, the cursor may have identified many rows in the ITEM table, but only the first row in the result set was processed. To process all rows in the result set, you may enclose the FETCH statement in a LOOP structure to repeatedly fetch and process each row until there were no more rows to process (R, 2015).

2.19 The *CURSOR FOR* Loop

The CURSOR FOR loop is used when you want to fetch and process every record in a cursor, and is terminated when all records in the cursor have been fetched. The loop can also be terminated using the EXIT statement. Once the loop body is executed, it performs another fetch. If the cursor returns no rows, then the body of the loop is never executed. It integrates procedural constructs with SQL, reduces the volume of code, and reduces the chance of loop errors (R, 2015).

The syntax of a cursor FOR loop:

```
FOR record_index IN cursor_name
LOOP
<Executable statement(s)>
END LOOP
```

Here's an example of how a cursor is used:

```
DECLARE
-- Declare variables
v_employee_id employees.employee_id%TYPE;
v_employee_name employees.first_name%TYPE;

-- Declare a cursor
CURSOR employee_cursor IS
```

```
SELECT employee_id, first_name
FROM employees
WHERE department_id = 30;

BEGIN
-- Open the cursor
OPEN employee_cursor;

-- Fetch and process each row using a loop
LOOP
    FETCH employee_cursor INTO v_employee_id, v_employee_name;

    -- Exit the loop if no more rows are to be fetched
    EXIT WHEN employee_cursor%NOTFOUND;

    -- Process the current row
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id || ', Employee Name: ' || v_employee_name);
END LOOP;

-- Close the cursor
CLOSE employee_cursor;

END;
```

In this example:

A cursor named `employee_cursor` is declared. This cursor retrieves the `employee_id` and `first_name` columns from the `employees` table for employees in department 30.

The cursor is opened using the `OPEN` statement.

A loop is used to fetch and process each row from the cursor result set. The `FETCH` statement retrieves the values into the declared variables `v_employee_id` and `v_employee_name`.

The loop continues until there are no more rows to fetch (`employee_cursor%NOTFOUND` is true).

Inside the loop, each row's information is printed using the `DBMS_OUTPUT.PUT_LINE` statement.

Finally, the cursor is closed using the `CLOSE` statement.

This example is a simple illustration of how to use a cursor in PL/SQL to fetch and process rows from a result set. Cursors are powerful constructs for working with query results in PL/SQL, especially when dealing with multiple rows of data.

An explicit cursor FOR loop is a loop that is associated with an explicit cursor or a SELECT statement incorporated directly within the loop boundary. When you use a FOR loop in cursors, you do not need to declare a record variable that holds cursor data. The OPEN, CLOSE, and FETCH steps are implicitly handled in a FOR loop cursor. Using a cursor FOR loop reduces the number of lines in the program

Syntax for the Explicit Cursor FOR Loop

The syntax to use a FOR loop with a cursor is:

```
FOR record_name IN cursor_name LOOP
```

```
statements...
```

```
END LOOP;
```

where record_name is a record declared implicitly by PL/SQL with the %ROWTYPE attribute against the cursor specified by cursor_name.

Explicit Cursor
FOR Loop

```
DECLARE
CURSOR cur1 IS SELECT first_name, job_id FROM employeeestable
WHERE job_id LIKE '%SALESMAN%' AND manager_id > 120;
BEGIN
FOR item IN cur1 LOOP
DBMS_OUTPUT.PUT_LINE ('Name = ' || item.first_name || ',
Job = ' || item.job_id);
END LOOP;
END;
/
```

Example of the Explicit Cursor FOR Loop

```
DECLARE
```

```
CURSOR cur1 IS SELECT first_name, salary FROM employeeestable WHERE salary
LIKE '%SALESMAN%' AND manager_id > 120;
```

```
BEGIN
```

```
FOR item IN cur1 /*cur1 will be stored in variable item*/
```

```
LOOP
```

```
DBMS_OUTPUT.PUT_LINE
```

```
('Name = ' || item.first_name || ', Job = ' || item.salary);
```

```
/*displays the first name and job ID of employees.*/
```

```
END LOOP;
```

```
END;
```

In the above example code, if the job_id is salesman and the manager_id is greater than 120, that employee's first name and job ID are displayed.

2.20 FOR UPDATE AND CURRENT OF

In Oracle PL/SQL, the FOR UPDATE and CURRENT OF clauses are used in conjunction with cursors to control and manipulate data during the processing of the cursor result set. Let's explore each of these clauses:

2.20.1 FOR UPDATE:

The FOR UPDATE clause is used in the SELECT statement when you want to lock the selected rows for update. This is often used in scenarios where you want to perform updates on the rows fetched by the cursor and want to ensure that other transactions cannot modify those rows until the current transaction is committed (R, 2015).

Here's an example of using FOR UPDATE:

```
DECLARE
  CURSOR employee_cursor IS
    SELECT employee_id, first_name
    FROM employees
    WHERE department_id = 30
    FOR UPDATE;

  v_employee_id employees.employee_id%TYPE;
  v_employee_name employees.first_name%TYPE;
BEGIN
  OPEN employee_cursor;
  FETCH employee_cursor INTO v_employee_id, v_employee_name;

  -- Perform some updates on the fetched row
  UPDATE employees
  SET salary = salary * 1.1
  WHERE CURRENT OF employee_cursor;

  -- Commit the changes to release the locks
  COMMIT;

  CLOSE employee_cursor;
END;
```

In this example, the FOR UPDATE clause is added to the SELECT statement in the cursor declaration. The UPDATE statement inside the loop then uses the WHERE CURRENT OF clause to specify the row to be updated based on the cursor's current position.

2.20.2 CURRENT OF:

The CURRENT OF clause is used in DML (Data Manipulation Language) statements to refer to the current row of the cursor. It allows you to update or delete the row currently pointed to by a cursor (R, 2015).

Here's an example of using CURRENT OF:

```
DECLARE
```

```
  CURSOR employee_cursor IS  
    SELECT employee_id, first_name  
    FROM employees  
    WHERE department_id = 30;
```

```
  v_employee_id employees.employee_id%TYPE;  
  v_employee_name employees.first_name%TYPE;
```

```
BEGIN
```

```
  OPEN employee_cursor;  
  FETCH employee_cursor INTO v_employee_id, v_employee_name;
```

```
  -- Perform some updates on the fetched row
```

```
  UPDATE employees  
  SET salary = salary * 1.1  
  WHERE CURRENT OF employee_cursor;
```

```
  -- Commit the changes  
  COMMIT;
```

```
  CLOSE employee_cursor;  
END;
```

```
DECLARE
```

```
  CURSOR employee_cursor IS  
    SELECT employee_id, first_name  
    FROM employees  
    WHERE department_id = 30;
```

```
  v_employee_id employees.employee_id%TYPE;  
  v_employee_name employees.first_name%TYPE;
```

```
BEGIN
```

```
  OPEN employee_cursor;  
  FETCH employee_cursor INTO v_employee_id, v_employee_name;
```

```
  -- Perform some updates on the fetched row
```

```
  UPDATE employees  
  SET salary = salary * 1.1  
  WHERE CURRENT OF employee_cursor;
```

```
-- Commit the changes  
COMMIT;
```

```
CLOSE employee_cursor;  
END;
```

In this example, the WHERE CURRENT OF clause is used in the UPDATE statement to specify that the update should be applied to the row currently pointed to by the employee_cursor.

Both FOR UPDATE and CURRENT OF are often used in scenarios where you want to perform updates based on the results of a cursor while ensuring the integrity of the data by locking the rows during the transaction. Always remember to commit or rollback your transaction to release the locks (R, 2015).

3 Practical Exercises

The following set of relations has been set up for a local events company. At present, the database is small and only includes information about artists, events and bookings. The relationships between the tables must be derived from the data within each of the tables.

The tables and the information required are as follows:

EVENT(EVENT_ID, EVENT_NAME, EVENT_RATE)

ARTIST(ARTIST_ID, ARTIST_NAME, ARTIST_EMAIL)

BOOKINGS(BOOKING_ID, BOOKING_DATE, EVENT_ID, ARTIST_ID)

Sample Data is shown below:

EVENT

EVENT_ID	EVENT_NAME	EVENT_RATE
1001	Open Air Comedy Festival	R 300
1002	Mountain Side Music Festival	R 280
1003	Beach Music Festival	R 195

ARTIST

ARTIST_ID	ARTIST_NAME	ARTIST_EMAIL
A_101	Max Trillion	maxt@isat.com
A_102	Music Mayhem	mayhem@ymail.com
A_103	LOL Man	lol@isat.com

BOOKINGS

BOOKING_ID	BOOKING_DATE	EVENT_ID	ARTIST_ID
1	15 July 2017	1002	A_101
2	15 July 2017	1002	A_102
3	27 August 2017	1001	A_103
4	30 August 2017	1003	A_101
5	30 August 2017	1003	A_102

- You will need to create the tables above to complete the queries. Please create the tables and populate them using SQL Developer or SQL*Plus.
- Create a PL/SQL query to display the artist name and the booking date for Event ID 1001. Sample output is shown below:

Sample Results

```
ARTIST NAME:  LOL Man
BOOKING DATE: 27/AUG/17
```

- Management of the events have decided to allow a 10% discount on all events that cost over R250. Create a PL/SQL query to display the event name and price, with or without the applied discount. Sample output is shown below:

Sample Results

Open Air Comedy Festival price: R 270

Mountain Side Music Festival price: R 252

Beach Music Festival price: R 195

Learning Unit 5: Using Variables and Lexical Units	
Learning objectives addressed in this learning unit: LO1: Compare the predefined data types; LO2: Discuss the scalar data types; LO3: Differentiate between the different types of large objects; LO4: Justify the use of user-defined PL/SQL subtypes; LO5: Create and use: <ul style="list-style-type: none"> • PL/SQL variables; and • PL/SQL constants. LO6: Distinguish between the different lexical units; LO7: Outline/summarise the PL/SQL character sets; LO8: Discuss delimiters; LO9: Explain what an identifier is; LO10: Justify why comments are added to promote readability and understanding; LO11: Compare the different literal types; LO12: Explain the purpose of the: <ul style="list-style-type: none"> • %TYPE attribute; • %ROWTYPE attribute. LO13: Create and use PL/SQL lexical units.	My notes

1 Introduction

So far in the course, you have written a simple PL/SQL program. But to write programs that enable you to perform data management tasks, you will need to use variables and lexical units within a code block. In this lesson, you will declare variables and use lexical units to perform basic data modifications.

Writing a program with a basic block structure will limit your programming capability. Using various PL/SQL variables and lexical units in a program will help you structure the program and use it to perform data operations. Using them, you will also be able to communicate between various blocks in a program, thus taking your program to the next level (R, 2015).

2 Use PL/SQL Variables and Data Types

In this lesson, you will declare variables and use lexical units to perform data operations. One common type of data manipulation is to transmit information within a PL/SQL program. This can be done by declaring variables and their associated data types in the declaration section. In this topic, you will identify PL/SQL variables and data types (R, 2015).

When you are writing a program, you may want to represent a value or an object. This is the point where variables come into action. PL/SQL lets you declare a variable, assign a descriptive name, and assign values to it, along with the data type, in the declaration section. You can then store data in memory as a variable, and, as you

execute a program, carry out various operations on that data. You can refer to the variable you have declared, reuse it, or even change it (R, 2015).

2.1 *Predefined Data Types*

Data types determine the storage format, valid range of values, constraints, and operations that can be performed on constants, variables, and parameters. PL/SQL provides predefined data types and subtypes, and also lets users define their own PL/SQL subtypes in the declaration section of a variable (R, 2015).

Oracle provides a host of pre-defined data types.

Data Type	Description
Scalar	Data items that are single values with no internal components. Examples of the scalar data types are “INTEGER” and “DECIMAL.”
Composite	Data items that have internal components that can be accessed individually. Examples of the composite data types are “RECORD” and “TABLE.”
Reference	Data items that point to other data items. Examples of the reference data types are “REFERENCE CURSOR” and “REFERENCE object_type.”
Large Object (LOB)	Data items that point to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms. Examples of the LOB data types are BLOB (Binary Large Objects) and CLOB (Character Large Objects).

2.2 *Scalar Data Types*

There are several predefined PL/SQL scalar data types that store specific kinds of data.

Scalar Data Type	Data Description
Number	Values that store fixed-point or floating-point numbers Examples of the number data type are '120' and '187.50.'
Character	Values that represent single characters or strings of characters, which you can manipulate. Examples of the character data type are 'c' and '5.'
BOOLEAN	Values using which you can perform logical operations. Examples of the boolean data type are 'AND' and '='.
Datetime	Values that let you store and manipulate dates and times. An Example of a date time data type is '16-05-2012.'
Interval	Time interval values using which you can manipulate data. An example of the interval data type is 'INTERVAL YEAR(3) TO MONTH.'

2.3 LOB Data Types

Large object (LOB) data types enable you to store blocks of unstructured data such as text, video clips, graphic images, and sound waveforms up to four gigabytes in size. This data type allows efficient and random piece-wise access to data.

Oracle provides several pre-defined LOB types (R, 2015).

LOB data type	Used to
BFILE	Store large binary objects in operating system files outside the database. A BFILE variable stores a file locator pointing to a large binary file on the server. The locator also includes a directory alias, which specifies a full path name. BFILEs are read-only and their size is system dependent but cannot exceed four gigabytes.
BLOB	Store large binary objects in the database, in-line or out-of-line. BLOBs can participate in transactions, are recoverable, and can be replicated. BLOB locators span transactions, but they cannot span sessions. The size of this data type cannot exceed 4 GB.
CLOB	Store large blocks of character data in the data- base, in-line or out-of-line. Both fixed-width and variable-width character sets are supported. CLOBs can participate in transactions, are recoverable, and can be replicated. CLOB locators span transactions, but they cannot span sessions. The size of this data type varies from 8 GB to 128 TB.
NCLOB	Store large blocks of NCHAR data in the database. Both fixed-width and variable-width character sets are supported. Every NCLOB variable stores a locator, which points to a large block of NCHAR data. NCLOBs participate fully in transactions, are recoverable, and can be replicated. The size of this data type varies from 8 GB to 128 TB.

2.3.1 LOB Locators

To refer to a large object that is stored in an external file, an LOB data type uses an LOB locator, which is stored in an external file, either inside the row (inline) or outside the row (out of line). In the external file the LOB locators are in columns of the types BFILE, BLOB, CLOB, and NCLOB. PL/SQL operates on large objects through their LOB locators (R, 2015).

For example, when you select a BLOB column value, PL/SQL returns only its locator. If PL/SQL returns the locator during a transaction, the locator includes a transaction ID, so you cannot use that locator to update that large object in another transaction. Likewise, you cannot save a locator during one session and then use it in another session (R, 2015).

2.4 User-Defined PL/SQL Subtypes

A subtype is a subset of another data type, which is called its base type. A subtype has the same valid operations as its base type, but only a subset of its valid values. Subtypes can increase reliability, provide compatibility with ANSI/ISO types, and improve readability by indicating the intended use of constants and variables.

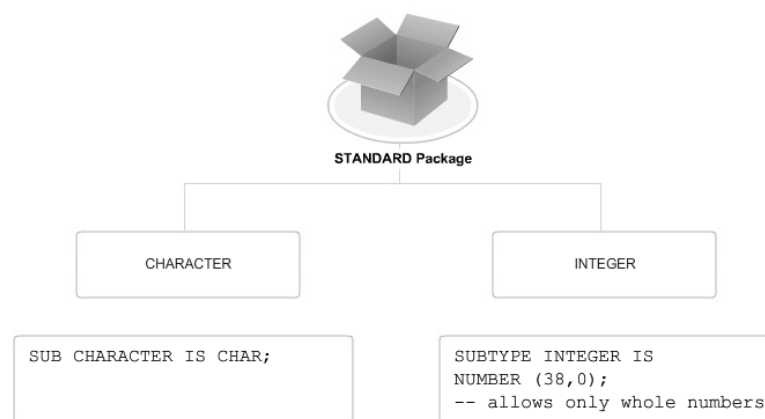
PL/SQL predefines several subtypes in the package STANDARD. For example, PL/SQL predefines the subtype CHARACTER as:

```
SUBTYPE CHARACTER IS CHAR;
```

and the subtype INTEGER as:

```
SUBTYPE INTEGER IS NUMBER(38,0); -- allows only whole numbers
```

The subtype CHARACTER specifies the same set of values as its base type CHAR, so CHARACTER is an unconstrained subtype. However, the subtype INTEGER specifies only a subset of the values of its base type NUMBER, so INTEGER is a constrained subtype (R, 2015).

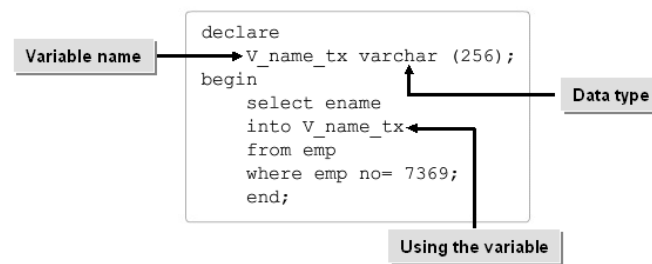


2.5 Declaring PL/SQL Variables

PL/SQL lets you declare variables, and then use them in SQL and procedural statements any- where an expression can be used. You must declare a variable before referring to it in any other statements. A PL/SQL variable can have any SQL data type (such as CHAR, DATE, or NUMBER) or a PL/SQL-only data type (such as BOOLEAN) (R, 2015).

The general syntax for declaring a variable is

variable_name data type [NOT NULL := value];



2.5.1 Assigning Values to Variables

You can assign a value to a variable in the following ways:

- With the assignment operator (**:=**).
- By selecting (or fetching) database values into it.
- By passing it as an OUT or IN OUT parameter to a subprogram, and then assigning the value inside the subprogram.

2.5.2 Bind Variables

Bind variables help to improve performance of a program by allowing the database to reuse SQL statements. When you embed a SQL INSERT, UPDATE, DELETE, or SELECT statement directly in your PL/SQL code, PL/SQL turns the variables in the WHERE and VALUES clauses into bind variables automatically. The database can reuse such SQL statements every time the same code is executed. To run similar statements with different variable values, you can save the parsing overhead by invoking a stored subprogram that accepts parameters and then issues the statements with the parameters substituted in the appropriate places (R, 2015).

2.5.3 PL/SQL Constants

A constant is a value used in a PL/SQL block that does not change throughout the program. A constant is a user-defined literal value. You can declare a constant and use it instead of an actual value. Declaring a PL/SQL constant is like declaring a PL/SQL variable except that you must add the keyword **CONSTANT** and immediately assign a value to the constant.

The general syntax of declaring a constant is
constant_name CONSTANT data type := VALUE;

For example: `credit_limit CONSTANT NUMBER := 5000.00;`
 No further assignments to the constant are allowed (R, 2015).

```

credit_limit
CONSTANT NUMBER := 5000.00;

```

2.5.4 Use PL/SQL Lexical Units

So far, you have identified the various variables and data types in PL/SQL. You now want to write programs with proper indentation. In this topic, you will use PL/SQL lexical units.

In learning any language, it is necessary that you first learn the keywords, what they mean, and how to use them. It is the same as when you are learning the PL/SQL language. Identifying and using the different types of lexical units will help you get the desired results (R, 2015).

2.6 Lexical Units

PL/SQL is not case-sensitive, so lower-case letters are equivalent to corresponding upper-case letters except within string and character literals (R, 2015). A line of PL/SQL text contains groups of characters known as lexical units, which can be classified as follows:

- Delimiters
- Identifiers
- Comments
- and, Literals

2.6.1 PL/SQL Character Sets

PL/SQL supports two character sets: the national character set, which is used for national language data, and the database character set, which is used for identifiers and source code (R, 2015).

PL/SQL programs are written as lines of text using the following characters:

- Upper- and lower-case letters
- Numerals
- Symbols
- and, Tabs, spaces, and carriage returns

2.6.2 Delimiters

A delimiter is a simple or compound symbol that has a special meaning to PL/SQL. There are several important delimiters.

Delimiter Symbol	Meaning
+	Addition operator. An example of a string using an addition operator is 'count := count + 1;.'
%	Percentage operator. An example of a string using a percentage operator is 'EXIT WHEN c1%NOTFOUND;.'
*	Multiplication operator. An example of a string

	using a multiplication operator is 'bonus := salary * 0.15;.'
=	An example of a string using a relational operator is 'WHERE empno = emp_id;.'
,	Item separator. An example of a string using an item separator is 'credit REAL(7,2);'
:=	Assignment operator. An example of a string using an assignment operator is 'counter := counter + 1;.'

2.6.3 Identifiers

Identifiers are used to name PL/SQL program items and units, which include constants, variables, exceptions, cursors, cursor variables, subprograms, and packages. The length of an identifier varies from one to 30 characters. The first character must be a letter, but the subsequent characters can either be a letter, numeral, dollar sign (\$), underscore (_), or number sign (#). PL/SQL is not case-sensitive with respect to identifiers. Every character, alphabetic or not, is significant (R, 2015).

2.6.4 Comments

Comments are added to your program to promote readability and understanding. Typically, comments are used to describe the purpose and use of each code segment. The PL/SQL compiler ignores comments. Users can also disable obsolete or unfinished pieces of code by turning them into comments.

There are two types of comments: single-line and multiline comment. A single-line comment begins with --. A multiline comment begins with /*, ends with */ and can span multiple lines (R, 2015).

```

declare
--used to declare variables, constants, etc.
begin
--used to place your code

```

Comment

2.6.5 Literals

A literal is an explicit numeric, character, string, or BOOLEAN value not represented by an identifier.

Literal Type	Description
Numeric Literal	Two types of numeric literals can be used in arithmetic expressions : integer and real. An integer literal is an optionally signed whole number with- out a decimal point. A real literal is an optionally signed whole or fractional number with a decimal point.
Character Literals	A character literal is an individual character enclosed by single quotes (''). Character literals include all printable characters in the PL/SQL character set: letters, numerals, spaces, and special symbols.
String Literals	A character value can be represented by an identifier or explicitly written as a string literal, which is a sequence of zero or more characters enclosed by single quotes. All string literals except for the null string ('') have the data type CHAR.
BOOLEAN Literals	BOOLEAN literals are the predefined values TRUE, FALSE, and NULL. NULL stands for a missing, unknown, or inapplicable value. Remember, BOOLEAN literals are values, not strings.
Date and Time Literals	Datetime literals have various formats depending on the data type.

Lexical units in PL/SQL are the basic building blocks of the language. They include identifiers, literals, keywords, and operators. Here's a brief overview of each:

Identifiers

An identifier is a name given to a variable, constant, exception, cursor, or other program objects.

It must start with a letter and can be followed by letters, numbers, or underscores.

Example: employee_name, total_sales, v_count.

Literals

A literal is a fixed data value used in a program.

Examples:

Numeric literals: 1, 3.14, -42

String literals: 'Hello, World!', 'John Doe'

Date literals: DATE '2023-11-23'

Character literals: 'A'

Keywords

Keywords are reserved words that have special meaning in PL/SQL and cannot be used as identifiers.

Examples: DECLARE, BEGIN, END, IF, LOOP, SELECT, UPDATE, DELETE, etc.

Operators

Operators perform operations on variables and values.

Examples:

Arithmetic operators: +, -, *, /

Comparison operators: =, !=, <, >

Logical operators: AND, OR, NOT

Assignment operator: :=

Here's a simple PL/SQL example that uses these lexical units:

DECLARE

-- Declaration section (identifiers)

v_employee_name VARCHAR2(50);

v_salary NUMBER := 50000; -- initialization using assignment operator

BEGIN

-- Execution section

v_employee_name := 'John Doe'; -- assignment statement

-- Conditional statement (keyword)

IF v_salary > 60000 THEN

 DBMS_OUTPUT.PUT_LINE('High salary for ' || v_employee_name);

ELSE

 DBMS_OUTPUT.PUT_LINE('Standard salary for ' || v_employee_name);

END IF;

-- Loop (keyword)

FOR i IN 1..3 LOOP

-- Output literal and identifier

 DBMS_OUTPUT.PUT_LINE('Iteration ' || i || ': ' || v_employee_name);

END LOOP;

END;

In this example:

DECLARE and BEGIN are keywords.

v_employee_name, v_salary, and i are identifiers.

'John Doe', 50000, 60000, and 3 are literals.

:=, >, ||, IF, ELSE, END IF, FOR, LOOP, and / are operators and control structures.

This is a basic example, and PL/SQL allows you to create more complex programs with procedures, functions, and exception handling. The above script, when executed in an Oracle environment, would display output using the DBMS_OUTPUT.PUT_LINE procedure.

2.7 The %TYPE Attribute

The %TYPE attribute provides data types of variables or database columns. This comes in handy while declaring variables that will hold database values. For example, assume there is a column named last_name in a table named employees. To declare a variable named v_last_name that has the same data type as the column last_name, use dot notation and the %TYPE attribute, as follows:

```
v_last_name employees.last_name%TYPE;
```

Declaring v_last_name with %TYPE has two advantages. First, you need not know the exact data type of last_name. Second, if you change the database definition of last_name, perhaps to make it a longer character string, the data type of v_last_name changes accordingly at runtime (R, 2015).

2.8 The %ROWTYPE Attribute

In PL/SQL, records are used to group data. A record consists of a number of related fields in which data values can be stored. The %ROWTYPE attribute provides a record type that represents a row in a table. The record can store an entire row of data selected from the table or fetched from a cursor or cursor variable. Columns in a row and corresponding fields in a record have the same names and data types (R, 2015). In the following example, you declare a record named dept_rec, whose fields have the same names and data types as the columns in the departments table:

```
dept_rec departments%ROWTYPE; -- declare record variable
```

3 Practical Exercises

Refer to the database we have done in the previous learning unit under practical exercises and do the query below:

Create a PL/SQL query to display the events that have an event price greater than the average price of all the events. Sample output shown below:

Sample Results

Open Air Comedy Festival

Price: R 300

Mountain Side Music Festival

Price: R 280

Learning Unit 6: Using Control Structures

Learning objectives addressed in this learning unit:

My notes

- LO1: Identify applicable/ suitable arithmetic operators used to perform expressions.
- LO2: Evaluate comparison operators to compare expressions or values.
- LO3: Explain how to combine two string expressions into one expression.
- LO4: Compare the three logical operators used in PL/SQL.
- LO5: Explain operator precedence.
- LO6: Create and use expressions.
- LO7: Contrast the types of Boolean expressions supported in procedural statements.
- LO8: Compare and formulate the three basic control structures.
- .
- LO9: Identify the types of IF statements used to control the execution of statements.
- LO10: Create and use CASE statements.
- LO11: Compare the three types of LOOP statements used in PL/SQL statements.
- LO12: Compare and use the two types of EXIT statements.
- LO13: Use the two forms of CONTINUE statements.

1 Introduction

So far, you have declared variables and used lexical units in a PL/SQL program to perform simple data calculations. Most real-life scenarios require that you follow a decision-making approach to solving problems. Using control structures will help you in decision making and branching of a program sequence. In this lesson, you will use control structures to help you incorporate a decision-making approach to programming.

To solve a mathematical calculation, you might decide either to use a calculator or to mentally solve it based on its complexity. Similarly, in programming, you can decide to run a different sequence of code depending on the result of a test condition to achieve the desired output (R, 2015).

Control structures provide an efficient way of checking the condition before an input is passed through code. Depending on the condition, the entry and exit points of a code block for a particular input are defined (R, 2015).

2 Use PL/SQL Operators and Expressions

In this lesson, you will use control structures to help you incorporate a decision-making approach to programming. One common example of this is when you have to write programs where you will need to calculate values to get a desired output. In this topic, you will create programming logic with PL/SQL operators and expressions (R, 2015).

When writing code, you will most definitely come across instances when you will need to compare, evaluate, and calculate values. Comparisons and calculations involve the use of expressions with appropriate operators and operands. Familiarity with the operators available in PL/SQL will allow you to create expressions and thereby use programming logic effectively (R, 2015).

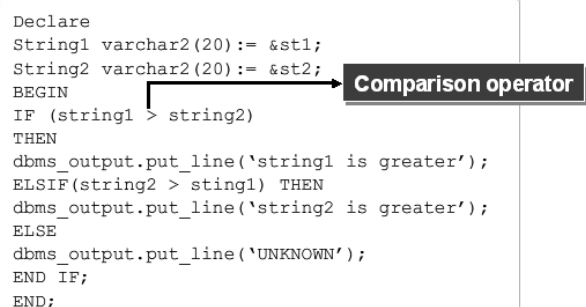
2.1 Arithmetic Operators

Arithmetic operators are operators used to perform mathematical operations on numeric expressions. However, the plus (+) and minus (-) operators can also be used to perform arithmetic operations on datetime and smalldatetime values. Frequently used arithmetic operators include the addition, subtraction, multiplication, division, and modular operators (R, 2015).

2.2 Comparison Operators

Comparison operators are symbols used to compare two expressions or values. The output of a comparison operator is one of three values: TRUE, FALSE, or UNKNOWN. Comparison operators cannot be used with text, ntext, or image data types. Some of the comparison operators include the equal to, greater than, less than, greater than or equal to, less than or equal to, and not equal to operators (R, 2015).

```
Declare
String1 varchar2(20):= &st1;
String2 varchar2(20):= &st2;
BEGIN
  IF (string1 > string2)
  THEN
    dbms_output.put_line('string1 is greater');
  ELSIF(string2 > sting1) THEN
    dbms_output.put_line('string2 is greater');
  ELSE
    dbms_output.put_line('UNKNOWN');
  END IF;
END;
```

A diagram showing a PL/SQL code block with a callout box. The code block contains a DECLARE section with two variables, String1 and String2, both of type varchar2(20), initialized with &st1 and &st2 respectively. The BEGIN section contains an IF statement: IF (string1 > string2) THEN dbms_output.put_line('string1 is greater'); ELSIF(string2 > sting1) THEN dbms_output.put_line('string2 is greater'); ELSE dbms_output.put_line('UNKNOWN'); END IF; END;. A callout box labeled 'Comparison operator' has an arrow pointing to the '>' operator in the IF statement condition.

2.3 The Concatenation Operator

The concatenation operator combines two string expressions into one string expression. If there are leading or trailing spaces in either of the expressions, then the expression which does not contain leading or trailing spaces is appended following the spaces. The + (String Concatenation) operator is used to concatenate two

expressions. Both expressions must be of the same data type, or one expression must be able to be implicitly converted to the data type of the other expression.

You can link specific columns to other columns, arithmetic expressions, or constant values to create a character expression by using the concatenation operator (||). Columns on either side of the operator are combined to make a single output column

. For example, the `LAST_NAME` and `JOB_ID` columns of the `emp_details` table can be concatenated and given the alias `employee_names` to make a single output column (R, 2015).

Example of Using the Concatenation Operator

The following SQL statement concatenates the `LAST_NAME` and `JOB_ID` columns and displays a single output column with the column alias as `EMPLOYEES`. The `AS` keyword makes the `SELECT` clause easier to read.

```
SELECT LAST_NAME||JOB_ID AS "EMPLOYEES"  
FROM employees;
```

2.4 Logical Operators

```
select firstname||lastname as "employee_names", 'incentives='||sal+2000 from
emp_details where no_of_days>=30;
```

Concatenation operator

A logical operator combines the result of two component conditions to produce a single result based on certain conditions or inverts the result of a single condition. Three logical operators can be used in PL/SQL.

Operator	Function
AND	<p>Returns TRUE if both component conditions are true. For example, when the following query is executed, both component conditions must be true for any record to be selected.</p> <pre>SELECT employee_id, last_name, job_id, salary FROM employees WHERE salary >= 10000 AND job_id LIKE '%MAN%';</pre> <p>Therefore, only those employees who have a job title that contains the string 'MAN' and earn \$10,000 or more are selected.</p>
OR	<p>Returns TRUE if either component condition is true. For example, when the following query is executed, either component condition can be true for any record to be selected:</p> <pre>.SELECT employee_id, last_name, job_id, salary FROM employees WHERE salary >= 10000 OR job_id LIKE '%MAN%';</pre> <p>Therefore, any employee who has a job ID that contains the string 'MAN' or earns \$10,000 or more is selected</p>
NOT	<p>Returns TRUE if the component condition is false. For example, when the following query is executed, the output displays the last names and job IDs of all employees whose job ID is not IT_PROG, ST_CLERK, and SA_REP.</p> <pre>SELECT last_name, job_id FROM employees WHERE job_id NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');</pre> <p>Therefore, any employee whose job ID is not IT_PROG, or ST_CLERK, SA_REP is selected</p>

2.5 Operator Precedence

When multiple operators are used in a complex expression, operator precedence determines the sequence in which operations are performed. A higher-level operator is evaluated before a lower-level operator. If the order of execution is not specified precisely using parentheses, the resulting output may not be correct.

Operators execute in order of their precedence levels. You can use parentheses to change the order in which operators are executed (R, 2015).

Operator Level	Operator Precedence in Each Level
1	+ (Positive), - (Negative), ~ (Bitwise NOT)
2	* (Multiply), / (Division), % (Modulo)
3	+ (Add), (Concatenate), - (Subtract)
4	=, >, <, >=, <=, <>, !=, !>, !< (Comparison operators)
5	^ (Bitwise Exclusive OR), & (Bitwise AND), (Bitwise OR)
6	NOT
7	AND
8	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
9	= (Assignment)

2.6 Expressions

An expression is a logical combination of operands and operators that can be evaluated to produce another value. An expression may or may not evaluate to a value depending on the correctness of the expression and depending on the operands that are used to create the expression. PL/SQL evaluates an expression by combining the values of the operands in ways specified by operators. An expression always returns a single value. PL/SQL examines the expression and the context in which it appears which determines the data type of this value (R, 2015).

2.7 BOOLEAN Expressions

PL/SQL lets you compare variables and constants in both SQL and procedural statements. These comparisons, called BOOLEAN expressions, consist of simple or complex expressions separated by relational operators. BOOLEAN expressions are often connected by the logical operators AND, OR, and NOT. A BOOLEAN expression always results in TRUE, FALSE, or NULL. In a SQL statement, BOOLEAN expressions let you specify the rows in a table that are affected by the statement. In a procedural statement, BOOLEAN expressions are the basis for conditional control (R, 2015).

```
if (sal>2000) AND (sal<5000)
```

```
if (sal>2000) OR (sal<5000)
```

```
if ((to_char(sysdate) not in ('mon')))
```

2.7.1 Types of BOOLEAN Expressions

Oracle supports three kinds of BOOLEAN expressions.

BOOLEAN Expression	Description
BOOLEAN Arithmetic Expressions	<p>Numbers are compared for equality or inequality using relational operators. Comparisons are quantitative; that is, one number is greater than another if it represents a larger quantity.</p> <p>For example, for the given assignments: variable1:= 80; variable2:= 75; the following expression variable1 > variable2 is true.</p>
BOOLEAN Character Expressions	<p>Character values can be compared for equality or inequality. By default, comparisons are based on the binary values of each byte in the string.</p> <p>For example, given the assignments: string1:= 'Kathy'; string2:= 'Kathleen'; the following expression string1 > string2 is true. Here the binary value of Kathy is greater than Kathleen.</p>
BOOLEAN Date Expressions	<p>Dates can also be compared. Comparisons are chronological; that is, one date is greater than another if it is more recent.</p> <p>For example, given the assignments date1:= '01-JAN-2010'; date2 := '31-DEC-2011'; the following expression date1 > date2 is false.</p>

2.7.2 Use Conditional Statements

In the first topic, you wrote programs using operators and expressions to sequentially run a code block and generate a specific output. There will be instances when you will want to include alternate execution logic into your code block as a condition to selectively branch and execute parts of a code block in a program to achieve specific results. In this topic, you will apply conditional statements (R, 2015).

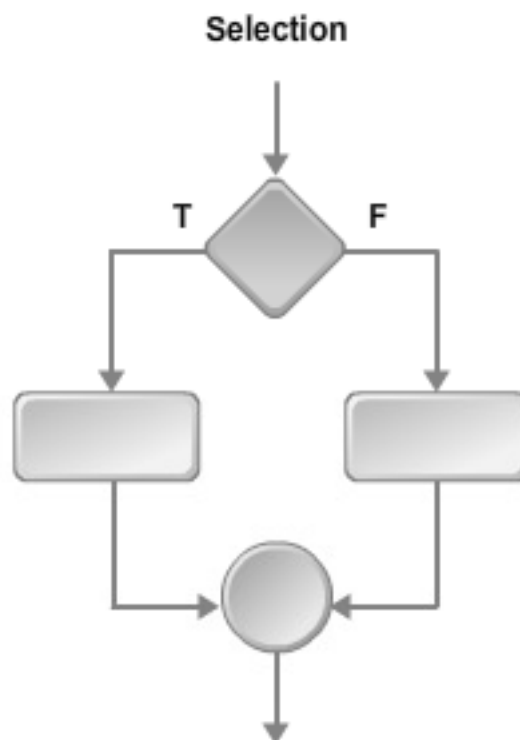
Real-time scenarios often require that programmers test a condition and execute different blocks of code to arrive at different results based on the result of the condition tested. This would require either creation of separate programs for each test outcome, or branching and selective execution of code within a program. Programs written using conditional statements are logically organized and easy to understand (R, 2015).

2.7.3 Control Structures in PL/SQL

Any computer program can be written using the basic control structures. They can be combined in any way necessary to deal with a given problem.

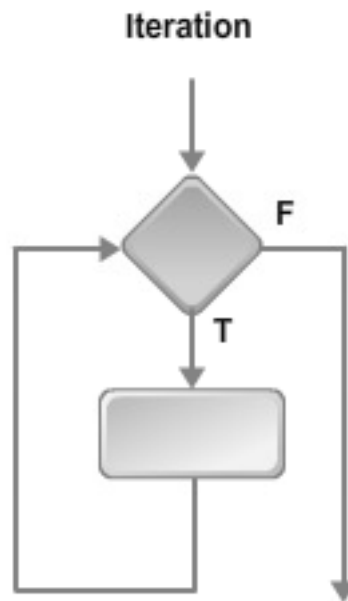
2.7.4 Selection Structure

Depending on whether the condition is true or false, the selection structure tests a condition, then executes one sequence of statements instead of another. A condition is any variable or expression that returns a BOOLEAN value (TRUE or FALSE).



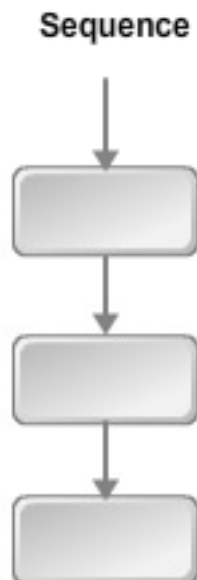
2.7.5 Iteration Structure

Executes a sequence of statements repeatedly as long as a condition is true.



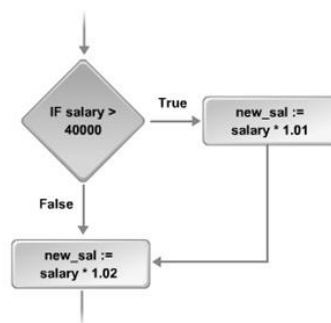
2.7.6 Sequence s

Executes a sequence of statements in the order in which they occur.



2.8 Conditional Control Statements

In almost every piece of code you write, conditional control is required. Conditional statements give you the ability to direct the flow of execution through your program based on a condition. You do this with IF and CASE statements. The IF statement lets you execute a sequence of statements conditionally, whether or not the sequence is executed depends on the value of a condition. Oracle supports three forms of IF statements: IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSIF. The CASE statement is a compact way to evaluate a single condition and choose between many alternative actions (R, 2015).



2.8.1 IF Statements

The IF statement is the most commonly used control-flow statement, and it executes a sequence of statements depending on the value of a condition. Oracle supports three forms of IF statements (R, 2015).

IF Statement	Description
IF-THEN	<p>The IF-THEN statement contains an expression to be evaluated and one or more actions to be performed, if the result of the expression is TRUE. If the expression is evaluated to FALSE, the actions are skipped, and the next program statement is executed. The syntax for the IF-THEN statement is:</p> <pre> IF condition THEN ..<<sequence_of_statements>>.. END IF ; </pre> <p>An example using the IF-THEN statement:</p> <pre> IF sales > quota THEN compute_bonus(empid); UPDATE payroll SET pay = pay + bonus WHERE empno = emp_id; END IF; </pre>

IF-THEN-ELSE	<p>The IF-THEN-ELSE statement allows a choice between two actions based on the evaluation of an expression. If the expression is TRUE, the first action is performed; otherwise, the action following the else block is performed. When more than one action is required, statements must be enclosed in braces.</p> <p>The syntax for the IF-THEN-ELSE statement is:</p> <pre>IF condition THEN <<sequence_of_statements1>>.. ELSE <<sequence_of_statements2>>.. END IF;</pre> <p>An example using the IF-THEN-ELSE statement:</p> <pre>IF sales>50000 THEN bonus:=1500; else bonus:=500; END IF;</pre>
IF-THEN-ELSIF	<p>To introduce additional conditions, where you can choose between several alternatives, the keyword ELSIF is used. If the first condition is FALSE or NULL, the ELSIF clause tests another condition. Any number of ELSIF clauses can be used within an IF statement. Conditions are evaluated one at a time from top to bottom. If any condition is TRUE, its associated sequence of statements is executed and control passes to the next statement. If all conditions are FALSE or NULL, the sequence in the ELSE clause is executed.</p> <p>The syntax for the IF-THEN-ELSIF statement is:</p> <pre>IF condition1 THEN <<sequence_of_statements1>>... ELSIF condition2 THEN <<sequence_of_statements2>>... ELSE <<<sequence_of_statements3>>>... END IF;</pre> <p>An example using the IF-THEN-ELSIF statement:</p> <pre>BEGIN IF sales > 50000 THEN bonus:= 1500; ELSIF sales > 35000 THEN bonus:= 500; ELSE bonus:= 100; END IF; END;</pre>

2.9 The CASE Statement

The CASE statement selects one sequence of statements to execute. It uses a selector to determine the sequence. A selector is an expression whose value is used to select one of several alternatives. Lengthy IF-THEN-ELSIF statements can be rewritten as CASE statements to make them more readable and efficient. The CASE statement begins with the keyword CASE (R, 2015).

followed by a selector. The selector expression is evaluated only once. The selector is followed by one or more WHEN clauses, which are checked sequentially. The clause is executed depending on the value of the selector. If the value of the selector equals the value of a WHEN clause expression, that WHEN clause is executed. The ELSE clause is optional (R, 2015).

```
CASE [ selector ] WHEN condition_1 THEN result_1
WHEN condition_2 THEN result_2 ...
WHEN condition_n THEN result_n
ELSE result
END
```

Example of the CASE Statement

Consider the code:

```
select emp_details, CASE owners
WHEN 'SYS' THEN 'The owner is SYS'
WHEN 'SYSTEM' THEN 'The owner is SYSTEM'
ELSE 'The owner is another value' END
from emp_details;
```

The selector expression owner is used to select one of the alternatives: SYS or SYSTEM. The selector expression is evaluated only once. The WHEN clause is executed depending on the value of the selector (R, 2015).

2.10 Using Iterative Statements

You now know how to alter the flow of instructions in a program based on a test condition. As a developer you will be faced with situations that will require that you write programs using iterative execution logic based on the result of a condition. Repetitive actions can be easily automated by using a program that has a loop built into the solution of the problem. In this topic, you will use control-flow looping statements to control the sequential flow of your instructions (R, 2015).

Suppose you need to print the details of employees stored in a database. You could access each record of the database and then print it, but this would involve a lot of coding, complicating the entire code. By using iterative statements, you can perform

such repetitive tasks and minimize code statements resulting in effective flow control for your application (R, 2015).

2.10.1 The LOOP Statement

A LOOP statement executes a sequence of statements for a specified number of times. PL/SQL provides three types of LOOP statements.

LOOP Statement	Description
Basic LOOP	<p>A basic LOOP can be used to execute a statement or set of statements until a condition is satisfied. In a basic loop, the condition upon which the loop ends is evaluated at the end of each iteration. Processing continues until the condition becomes true. The loop counting variable must be declared and initialized before the loop body. It is incremented only within the loop body. An EXIT condition must be specified, or the loop will go into an endless number of iterations. When the EXIT condition is met, the program exits the loop. If the EXIT statement is used without the WHEN condition, the statements in the loop are executed only once.</p> <p>The syntax to write a basic loop is:</p> <pre> LOOP ..<<set of statements>>... EXIT WHEN<<conditionA>>; ..<<set of statements>>... END LOOP; An example of the basic loop is: DECLARE i number:= 1; BEGIN LOOP EXIT WHEN i>10; dbms_output.put_line(i); i:= i+1; END LOOP; END; </pre>
WHILE LOOP	<p>A WHILE LOOP is used when a sequence of statements has to be executed as long as a condition is true. In a WHILE LOOP the condition is evaluated at the beginning of each iteration. Processing continues until the condition becomes false. The loop-counting variable must be declared and initialized before the loop body. It is incremented only within the loop body.</p> <p>The syntax to write a WHILE LOOP is:</p> <pre> WHILE <<condition>> LOOP ..<<statements>>... END LOOP; An example of a while loop is: DECLARE i integer:= 1; BEGIN WHILE i < 10 LOOP dbms_output.put_line(i); i := i+1; END LOOP; END; </pre>

FOR LOOP	<p>A FOR LOOP is used to execute a sequence of statements a pre-determined number of times. No condition is evaluated, whether at the beginning or end of each iteration. Processing occurs between the starting and ending integers of the loop counting variable, which can be any expression that yields an integer value. This expression will be evaluated only once, at the start of the loop. The loop-counting variable is self-declared, known only in the loop and always incremented by 1.</p> <p>The syntax to write a FOR LOOP is:</p> <pre>FOR <counter> in <lower_bound>..<higher_bound> LOOP ...<<statements>>... END LOOP;</pre> <p>An example of a for loop is:</p> <pre>BEGIN FOR k in 1..10 LOOP dbms_output.put_line(k); END LOOP; END;</pre>
----------	--

2.11 EXIT Statements

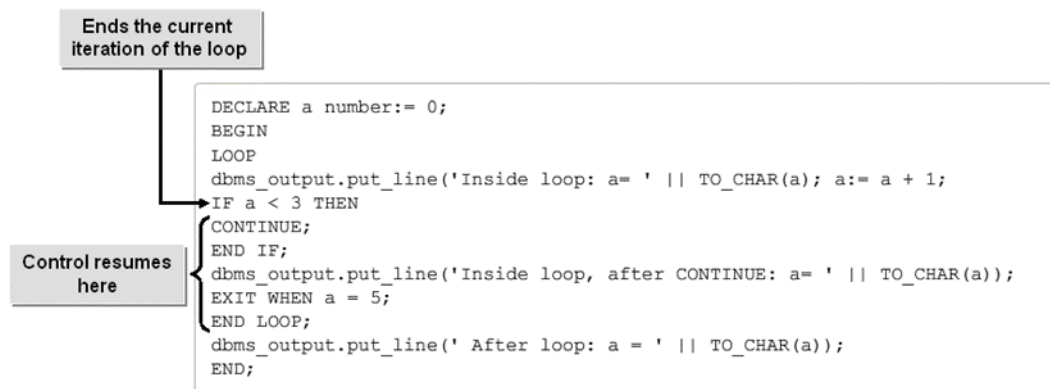
One or more EXIT statements can be used anywhere inside a loop, but nowhere outside a loop. Oracle supports two types of EXIT statements.

Exit Statement	Description
EXIT	<p>The EXIT statement forces a loop to complete unconditionally. When an EXIT statement is encountered, the loop stops immediately and the control passes to the next statement.</p> <p>An example of an EXIT statement:</p> <pre>LOOP IF credit_rate<4 THEN EXIT; -- exit loop immediately END IF; END LOOP; -- control resumes here</pre>
EXIT-WHEN	<p>The EXIT-WHEN statement allows a loop to complete conditionally. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. The loop completes if the condition is true and control passes to the next statement after the loop. The loop cannot complete, until the condition is true. Until the condition is true, the loop cannot exit. So, a statement within the loop must change the value of the condition. The EXIT-WHEN statement can be used to replace an IF statement.</p> <p>For example, compare the following statements:</p> <pre>IF count>100 THEN EXIT WHEN count > 100; EXIT;</pre>

	END IF; These statements are logically equivalent, but the EXIT-WHEN statement is easier to read and understand.
--	---

2.12 CONTINUE Statements

A CONTINUE statement ends the current iteration of a loop. This statement ends the processing of the action part of an iterative statement and moves control to the loop continuation portion of the statement. Oracle supports two forms of CONTINUE statements (R, 2015).



CONTINUE Statement	Description
CONTINUE	<p>When a CONTINUE statement is encountered, the current iteration of the loop completes immediately and control passes to the next iteration of the loop. You can name the loop to be exited. An example of the CONTINUE statement:</p> <pre> DECLARE a number := 0; BEGIN LOOP -- After CONTINUE statement, control resumes here dbms_output.put_line('Inside loop: a = ' TO_CHAR(a)); a := a + 1; IF a < 3 THEN CONTINUE; END IF; dbms_output.put_line('Inside loop, after CONTINUE: a = ' TO_CHAR(a)); EXIT WHEN a = 5; END LOOP; dbms_output.put_line(' After loop: a = ' TO_CHAR(a)); END; </pre>

CONTINUE-WHEN	<p>When a CONTINUE-WHEN statement is encountered, the condition in the WHEN clause is evaluated. The current iteration of the loop completes, if the condition is true, and control passes to the next iteration. The CONTINUE-WHEN statement acts like a NULL statement and does not terminate the iteration until the condition is true.</p> <p>An example of the CONTINUE-WHEN statement:</p> <pre>DECLARE a NUMBER:= 0; BEGIN LOOP -- After CONTINUE statement, control resumes here dbms_output.put_line('Inside loop: a= ' TO_CHAR(a)); a:= a + 1; CONTINUE WHEN a < 3; dbms_output.put_line('Inside loop, after CONTINUE: a= ' TO_CHAR(a)); EXIT WHEN a= 5; END LOOP; dbms_output.put_line(' After loop: a= ' TO_CHAR(a); END;</pre>
---------------	---

3 Practical Exercises

You have recently been hired by a local pet shop to assist with their database and data management. In your feedback to Top Management, you have motivated for the shop to use Oracle as a database Management System (R, 2015).

The following set of relations has been set up for the pet shop. At present the database is small and only includes information about customers, pets and pet sales. The relationships between the tables must be derived from the data in each of the tables.

The tables and the information required are as follows:

CUSTOMERS (CUSTOMER_ID, FIRST_NAME, SURNAME, EMAIL)

PETS (PET_ID, PET_TYPE, PET_PRICE, PET_STOCK_LEVEL)

PET_SALES (SALES_ID, SALES_DATE, SALES_QUANTITY, PET_ID, CUSTOMER_ID)

Sample data is shown below:

CUSTOMERS

CUSTOMER_ID	FIRST_NAME	SURNAME	EMAIL
1001	Patrick	Smith	ps@yahoo.com
1002	Steven	Hewson	shew@gmail.com
1003	Barry	Goodwin	barry@isat.co.za

PETS

PET_ID	PET_TYPE	PET_PRICE	PET_STOCK_LEVEL
111	Dog - Jack Russel	900	5
112	Cat - Persian	700	3
113	Bird - Budgie	190	8

PET_SALES

SALES_ID	SALES_DATE	SALES_QUANTITY	PET_ID	CUSTOMER_ID
Sale_101	15 March 2020	2	111	1003
Sale_102	17 March 2020	1	112	1003
Sale_103	19 March 2020	1	112	1001
Sale_104	20 March 2020	3	113	1002

You will need to create the above tables to complete the queries below. Please create the tables and populate them using SQL Developer or SQL*Plus.

Create a PL/SQL query to display the customer's name and the pet that was purchased. In your solution only display the pet sale quantity greater than 1. Sample output is shown below:

Sample Output

CUSTOMER NAME: Barry Goodwin

PET TYPE: Dog - Jack Russel

SALE QUANTITY 2

CUSTOMER NAME: Steven Hewson

PET TYPE: Bird - Budgie

SALE QUANTITY 3

Create a PL/SQL query to display the customer name and total amount spent by each customer. In your solution determine the customer rating. If the total amount spent is greater than or equal to R 1000, the customer receives a star rating, otherwise no star rating applies.

Sample Output

FIRST NAME: Patrick

SURNAME: Smith

AMOUNT: R 700

FIRST NAME: Steven

SURNAME: Hewson

AMOUNT: R 570

FIRST NAME: Barry

SURNAME: Goodwin

AMOUNT: R 2500 (***)

Learning Unit 7: Handling PL/SQL Exceptions

Learning objectives addressed in this learning unit:

My notes

LO1: Explain how to deal with exceptions.

LO2: Explain the rules for declaring exceptions.

LO3: Discuss the advantages of exceptions.

LO4: Justify the use of pre-defined system exceptions.

LO5: Explain how to handle user-defined exceptions.

LO6: Describe how to deal with unhandled exceptions.

LO7: Create PL/SQL blocks with exception handling.

1 Introduction

You used control structures to enhance the decision-making process to improve programming capability. In order to deal with programming errors that are likely to occur and pass clear messages to the user about the exact nature of the error that has occurred, you will declare, raise, and handle exceptions (R, 2015).

Coding errors are common in programming, and without a way to deal with them, they can cause an application to quit abruptly. Providing an error message can help users solve the problem, but the message must be clear and understandable to be effective. Knowing how to declare, raise, and handle exceptions in a PL/SQL program will help to mitigate programming issues (R, 2015).

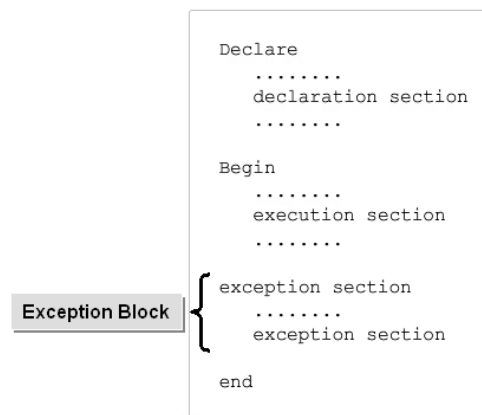
2 Handling System-Defined Exceptions

You used conditional logic statements to control the flow of your instructions, thus gaining programming knowledge. Errors are a common occurrence in any programming language. You now want to use the exceptions Oracle provides in its database, to control these errors. In this topic, you will equip yourself to handle system-defined exceptions (R, 2015).

As a programmer, you might be facing a host of issues, some of which are repeated frequently. In such cases, you would rather use pre-defined troubleshooting techniques, instead of treating each problem individually. In Oracle PL/SQL, there are some exceptions which need to be raised frequently. In order to not define them each time they are pre-defined and given a name to make them easier to use (R, 2015).

2.1 Exceptions

In PL/SQL programs, errors are referred to as exceptions. PL/SQL raises an exception for a system generated error or an error caused by the user. Sometimes exceptions can also be warnings the application issues to the user. When an exception is raised, the execution of the current block stops and the control transfers to the exception section. However, after handling the exception, the control doesn't go back to the block whose execution halted (R, 2015).



2.2 Exception Handling

Exception handling is a mechanism that enables you to deal with runtime errors caused by running a PL/SQL program. Exception handlers are located in the exception section and consist of a sequence of error-processing statements to trap and handle exceptions. If you have nested PL/SQL blocks, the code handling the exception stops and the overall processing continues in the next outer block. In fact, if the current block does not handle the exception, Oracle looks for an appropriate exception handler in subsequent blocks until it reaches the outermost block. If the exception has not been handled by that time, the entire program will stop abruptly. Oracle provides two types of exceptions: pre-defined exceptions and user-defined exceptions.

The sequence in which exception handling occurs is:

1. Declaring the exception.

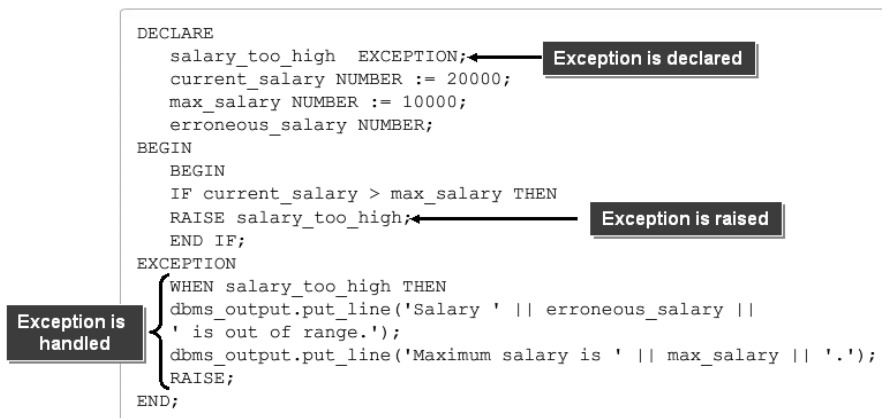
By declaring the exception, you're creating a new type of exception. Many Oracle predefined exceptions also exist, and you don't need to explicitly declare them.

2. Raising the exception.

After the exception is declared, it can be raised within the program. For user-defined exceptions, you raise the exception in response to something that happens in your program. For predefined exceptions, the exceptions are automatically raised when the problem occurs.

3. Handling the exception.

After an exception has occurred, the program stops normal execution and control is transferred to a special part of the program unit called the exception handler.



2.2.1 Scope of an Exception

There are a few rules that you need to follow while declaring an exception.

- A similar exception can be declared in two different blocks but cannot be declared twice within the same block.
- Exceptions declared in a block are considered local to that block and global to all its sub-blocks.
- A block can reference only local or global exceptions, therefore enclosing blocks cannot reference exceptions declared in a sub-block.
- If you redeclare a global exception in a sub-block, the local declaration prevails.

2.2.2 Advantages of Exceptions

Using exceptions as a means to handle errors has several advantages.

- Potential errors from many statements can be handled by using a single exception handler.
- Instead of troubleshooting for an error at every probable instance, adding an exception handler to the block will handle an error in the event that the exception being raised is in that block.
- Keeping the error-handling routines separate makes the rest of the program easier to read and understand.
- Explore in detail the various stages of the entrepreneurial lifecycle and how the entrepreneur can apply each stage of the lifecycle to their advantage.

2.2.3 Pre-Defined System Exceptions

System-defined exceptions occur whenever a program violates a rule of the RDBMS and are raised automatically.

Exception	Occurs
COLLECTION_IS_NULL (ORA-06531)	When you use a collection method (except EXISTS) on an uninitialized PL/SQL collection.
CURSOR_ALREADY_OPEN (ORA-06511)	When you open a cursor that is already open.
DUP_VAL_ON_INDEX (ORA-00001)	When you INSERT non-unique data in a column with the UNIQUE or PRIMARY KEY constraint.
NO_DATA_FOUND (ORA-01403)	When a SELECT...INTO does not return any row, or you reference a deleted element in a nested table.
PROGRAM_ERROR (ORA-06501)	When an unknown internal error occurs in PL/SQL.
STORAGE_ERROR (ORA-06500)	When your system runs out of memory.
ZERO_DIVIDE (ORA-06500)	When you attempt to divide a number by zero.

2.3 Handling User-Defined Exceptions

In the previous topic, you used system-defined exceptions to control commonly occurring programming errors. However, there might be instances where system-defined exceptions might not cater to your specific requirement. In this topic, you will define your own exceptions within a PL/SQL program.

Using system-defined exceptions may be useful for frequently occurring programming errors. However, in cases where the requirement for the exception is very specific, defining the exception to meet user requirements is beneficial (R, 2015).

2.3.1 User-Defined Exceptions

PL/SQL lets users define exceptions on their own. Unlike predefined exceptions, user-defined exceptions must be declared and then raised explicitly, using either a procedure or a RAISE statement. The latter lets you associate an error message with a user-defined exception. The general syntax for the RAISE statement is RAISE_APPLICATION_ERROR (error_number, error_message); Exceptions can be declared only in the declarative part of a PL/SQL block, subprogram, or package. You declare an exception by introducing its name, followed by the keyword EXCEPTION (R, 2015).

```
DECLARE
  Child_rec_exception EXCEPTION;
  PRAGMA
    EXCEPTION_INIT (Child_rec_exception, -2292);

BEGIN
  Delete FROM product where product_id= 104;

EXCEPTION
  WHEN Child_rec_exception
  THEN Dbms_output.put_line
    ('Child records are present for this product_id.');
```

```
END;
/
```

2.3.2 Unhandled Exceptions

If an exception is raised in a program and that exception is not handled by an exception section in either the current or subsequent PL/SQL blocks, that exception is unhandled.

Unhandled exceptions during the execution of a bulk operation cause the entire operation to be rolled back. A well-designed application will not allow unhandled exceptions to occur. The best way to avoid unhandled exceptions is to make sure that the outermost PL/SQL block contains a WHEN OTHERS clause in its exception section (R, 2015).

In Oracle PL/SQL, you can create user-defined exceptions to handle specific error conditions in your code. Here are some examples of how you can create and use user-defined exceptions:

Example 1: Simple User-Defined Exception

```
DECLARE
  -- Declare a user-defined exception
  custom_exception EXCEPTION;

BEGIN
  -- Raise the user-defined exception
  RAISE custom_exception;

EXCEPTION
  -- Catch the user-defined exception
  WHEN custom_exception THEN
    DBMS_OUTPUT.PUT_LINE('Custom Exception Caught');
END;
```

Example 2: User-Defined Exception with Error Message

```
DECLARE
  -- Declare a user-defined exception with an error message
  custom_exception EXCEPTION;
  PRAGMA EXCEPTION_INIT(custom_exception, -20001);

BEGIN
  -- Raise the user-defined exception with a custom error message
  RAISE_APPLICATION_ERROR(-20001, 'This is a custom error message');

EXCEPTION
  -- Catch the user-defined exception
  WHEN custom_exception THEN
    DBMS_OUTPUT.PUT_LINE('Custom Exception Caught: ' || SQLERRM);
END;
```

Example 1: A simple user-defined exception is declared and raised, and then caught in the exception block.

Example 2: A user-defined exception is declared with a custom error message using `RAISE_APPLICATION_ERROR`, and the exception is caught with the error message.

3 Practical Exercises

Create 3 PL/SQL queries that have user-defined exceptions that were discussed in this Learning Unit.

Learning Unit 8: Creating PL/SQL Subprograms

Learning objectives addressed in this learning unit:

LO1: Motivate the use of stored subprograms.
 LO2: Describe the parts of a subprogram.
 LO3: Compare the benefits and uses of subprograms.
 LO4: Compare anonymous blocks to subprograms.
 LO5: Create and use:

- Stored procedures;
- Return statements.

LO6: Compare functions and procedures.
 LO7: Create and use stored functions.
 LO8: Compare the different categories of built-in functions.
 LO9: Compare implicit and explicit conversions.
 LO10: Explain what overloading is.
 LO11: Compare the different subprogram parameter modes.
 LO12: Discuss the NOCOPY hint.
 LO13: Describe the function of the DETERMINISTIC clause.
 LO14: Create and execute simple triggers.
 LO15: Compare the different:

- Trigger categories;
- Trigger states.

LO16: Create and use compound triggers.
 LO17: Distinguish between the various compound timing points.
 LO18: Explain trigger execution.

My notes

1 Introduction

You have used control-flow structures to control the execution logic of instructions in a program. Dividing large, complex PL/SQL programs into multiple subprograms helps you stay focussed on a particular task, increasing programming efficiency. In this lesson, you will create PL/SQL subprograms.

When you have to solve a complex mathematical problem, you can complete computing easily if you break the problem into multiple smaller problems and independently solve them to arrive at the overall solution. Similarly, as a developer when you have to write large programs that enable complex computations, it is best if you break your programming task into subtasks performed within smaller program modules. By writing code for these smaller modules as independent programs, you can also increase program efficiency and reusability (R, 2015).

2 Use Stored Subprograms

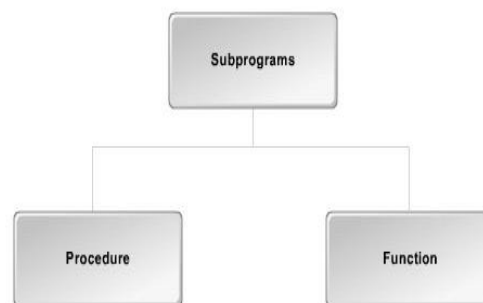
You have used cursors and exceptions in your programs. There will be instances when certain blocks of code in a program are similar and you can avoid additional effort by reusing or calling them at different instances in the large program. In this topic, you will use stored subprograms (R, 2015).

There will be instances where you will want to store a particular set of code and repeatedly want to execute it. PL/SQL provides you with a concept called stored subprograms where you can store compiled programs in Oracle's data dictionary as a database object which can be re-executed. Using subprograms will help break a program into manageable, well-defined modules, which in turn helps extend the PL/SQL language (R, 2015).

2.1 Subprograms

PL/SQL subprograms are named PL/SQL blocks that can be called with a set of parameters. A subprogram can either be a procedure or a function. Procedures are used to perform actions and functions to calculate and return a value. Subprograms can be created either at the schema level, inside a package, or inside a PL/SQL block (R, 2015).

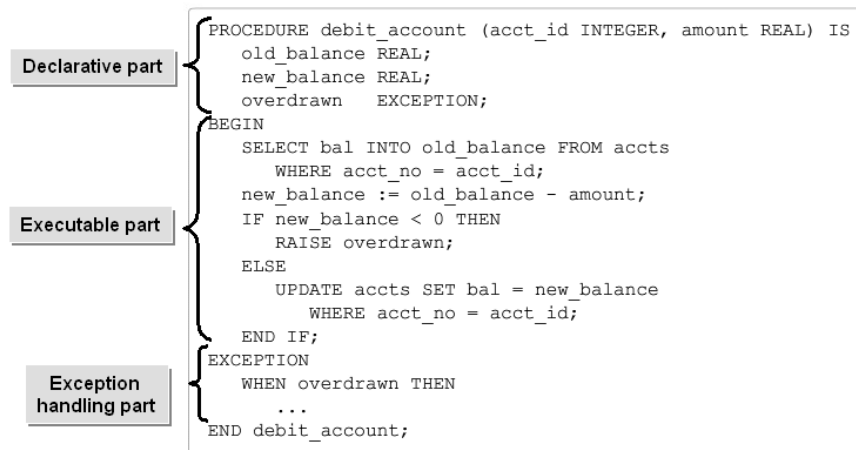
A subprogram created at the schema level is a standalone stored subprogram. A subprogram created inside a package is a packaged subprogram. A subprogram created inside a PL/SQL block is a nested subprogram. You create it with the CREATE PROCEDURE or CREATE FUNCTION statement.



2.1.1 Subprogram Parts

A subprogram is a program that has a declarative part, an executable part, and an optional exception-handling part. The declarative part of a subprogram does not begin with the keyword DECLARE, as the declarative part of a non-subprogram block does. The declarative part contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms (R, 2015).

These items are local to the subprogram and cease to exist when the subprogram completes execution. The executable part of a subprogram contains statements that assign values, control execution, and manipulate data. The exception-handling part of a subprogram contains code that handles runtime errors (R, 2015).



2.1.2 Benefits of Subprograms

You can use subprograms because they offer you real-time benefits such as modularity, efficiency, reusability, and maintainability.

- **Modularity:** A subprogram lets you split programs into well-defined, manageable modules. Either the top-down design or the stepwise refinement approach can be adopted for problem solving.
- **Re-usability:** Subprograms promote re-usability. Once tested, a subprogram can be reused in a number of applications. PL/SQL subprograms can also be invoked at several instances within a program or even from different environments, so there is no need to rewrite them each time a new language or API is being used.
- **Maintainability:** Subprograms promote maintainability. The internal details of a subprogram can be edited without disturbing other subprograms that invoke it. Subprograms are an important component of other maintainability features, such as packages and object types.
- **Dummy subprograms** defer definition of procedures and functions until after the main program has been tested.

2.2 Anonymous Blocks vs. Subprograms

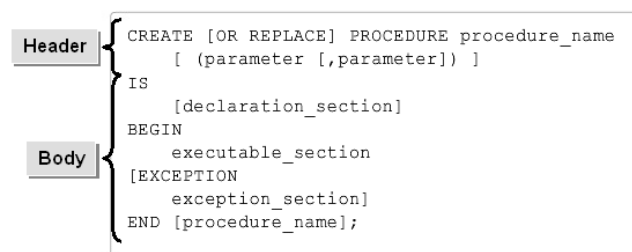
Anonymous blocks can also be used as nested blocks inside procedures, functions, and other anonymous blocks. Anonymous blocks are also PL/SQL blocks, but they are not named.

Therefore, these are not stored on the server for further reference. A subprogram is a named PL/SQL block, will be stored on the server and is available for future reference.

Anonymous blocks are stored on the client while subprograms are stored on the server (R, 2015).

2.3 Stored Procedures

A procedure is a named PL/SQL block that performs one or more specific tasks. Stored procedures are stored in the database data dictionary. A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists of a declaration section, an execution section and an exception section similar to a general PL/SQL block. A procedure is similar to an anonymous PL/SQL block but it is named for repeated usage. You can use nested stored procedures, by executing one stored procedure from within another (R, 2015).



2.3.1 The CREATE PROCEDURE Statement

The CREATE PROCEDURE statement creates or replaces a standalone stored procedure or a call specification. There can be any number of parameters, each followed by a mode and a type. You can also use the SQL CALL statement to call such a method or routine. The call specification tells the database which Java method, or which named procedure in which shared library, to invoke when a call is made. It also tells the database what type conversions to make for the arguments and return value (R, 2015).

```

CREATE
[OR REPLACE]
PROCEDURE procedure_name
[ (parameter [,parameter]) ]

```

2.3.2 Return Statements

The RETURN statement immediately completes the execution of a subprogram and returns control to the caller. Execution then resumes with the statement following the subprogram call. A subprogram can contain several RETURN statements. The last lexical statement does not need to be a RETURN statement. Executing any RETURN statement completes the subprogram immediately. In procedures, a RETURN statement cannot return a value, and therefore cannot contain an expression.

However, in functions, a RETURN statement must contain an expression, which is evaluated when the RETURN statement is executed (R, 2015).

```
CREATE OR REPLACE FUNCTION employer_details_func
    RETURN VARCHAR(20);
IS
    emp_name VARCHAR(20);
BEGIN
    SELECT first_name INTO emp_name
    FROM emp_tbl WHERE empID = '100';
    RETURN emp_name;
END;
```

2.4 Functions

Definition:

A function is a named PL/SQL block which you use to compute a value. It accepts parameters and always returns a value to the calling environment. A function should have a RETURN clause in the header and at least one RETURN statement in the executable section (R, 2015).

```
CREATE [OR REPLACE] FUNCTION function_name
[parameters]
RETURN return_datatype;
IS
Declaration_section
BEGIN
Execution_section
Return return_variable; EXCEPTION
exception_section
Return return_variable;
END;
```

Example:

```
CREATE OR REPLACE FUNCTION emp_details_func RETURN VARCHAR(20);
IS
emp_name VARCHAR(20); BEGIN
SELECT first_name INTO emp_name FROM emp_tbl WHERE empID = '50'; RETURN
emp_name;
END;
```

2.4.1 Uses of Functions

The following are the advantages of using functions in PL/SQL code (R, 2015):

- PL/SQL consists of blocks of code, which can be nested within each other. Each block forms a unit task or a logical module.
- PL/SQL blocks are stored in the database and can be reused.

- PL/SQL provides procedural language compatibility as it consists of conditional statements (if else statements) and loops (FOR loops).
- The PL/SQL engine enhances the performance by processing multiple SQL statements simultaneously as a single block, reducing network traffic.
- PL/SQL improves error handling capacity by handling errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of exception or it can be displayed to the user with a message.

2.5 Functions vs. Procedures

The differences between procedures and functions are listed below (R, 2015):

- Functions are mainly used when a value is to be returned. A procedure may or may not return a value or may return more than one value.
- Functions can be called from SQL statements whereas procedures cannot be called from SQL statements.
- Functions are normally used for computations whereas procedures are normally used for executing business logic.
- Functions return only a single value, whereas procedures can return multiple values up to 1024.
- A function return type could be scalar, table, or table values. Stored procedures always return integer values by default zero.
- Functions also contain DML (insert, update, delete) statements, but such a function cannot be called in a SQL query.

2.6 Stored Functions

If you want to reuse a function multiple times, it is easier to do so if you store the function in the database as a schema object (R, 2015). You can call such stored functions from within a SQL expression using the syntax

```
[schema_name.][pkg_name.][func_name[@db_link_name]][parameter_list]
```

where `schema_name` is the optional name of the schema in which the function is defined. It is usually your own Oracle account.

`pkg_name` is the optional name of the package in which the function is defined,

`func_name` is the name of the function,

`db_link_name` is the optional name of the database link if you are executing a remote procedure call, and

`parameter_list` is the optional list of parameters for the function.

2.7 Built-In PL/SQL Functions

Built-in SQL functions are built into the Oracle database and are available for use in various appropriate SQL statements. Users are provided with many powerful functions that help them manipulate data. They can be used where expressions of the same type are allowed. They can be nested as well (R, 2015).

Built in functions will fall under the following categories:

- Character functions
- Conversion functions
- Number functions
- Error-reporting functions
- Data functions

2.8 Create User-Defined Subprograms

You have used simple stored subprograms which are present by default in the Oracle database. In order to travel beyond the scope of the Oracle provided framework, PL/SQL provides you with the flexibility to create your own subprograms. In this topic, you will create your own subprograms (R, 2015).

Working with subprograms that are already present in the database may restrict your programming freedom. PL/SQL lets you define your own subprograms, extending flexibility and functionality (R, 2015).

2.9 Implicit Conversion

Implicit conversion is when PL/SQL can convert a value from one data type to another automatically. The data types that can be converted this way are called compatible. When two data types are compatible, you can use a value of one type where a value of the other type is expected. For example, you can pass a numeric literal to a subprogram that expects a string value, and the subprogram receives the string representation of the number (R, 2015).

2.10 Explicit Conversion

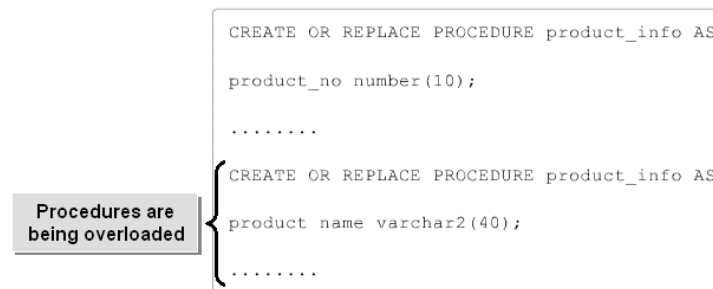
To explicitly convert values from one data type to another, you use built-in functions. For example, to convert a CHAR value to a DATE or NUMBER value, you use the function TO_DATE or TO_NUMBER, respectively. Conversely, to convert a DATE or NUMBER value to a CHAR value, you use the function TO_CHAR (R, 2015).

2.11 Overloading

Overloading is a process of creating numerous programs in a package each with an identical name. The main advantage of overloading is that it makes applications more user-friendly. If a procedure is present that performs the same task for the NUMBER,

CHAR, and VARCHAR2 parameters, you can overload procedures instead of creating separate ones for each data type(R, 2015).

For example, if you want to write procedures to display product information (either the name or the unit_price), you can create two overloaded procedures within a package. Both procedures can have the same name such as product_info, while differing only in input parameters. So, as far as the user is concerned, what is seemingly the same procedure accepts different types of parameters (R, 2015).



2.12 Subprogram Parameter Modes

Subprogram parameter modes are used to define the behavior of formal parameters. The three parameter modes are IN, OUT, and IN OUT, with the default being IN. The IN parameter passes a value to a subprogram that is being called. Inside the subprogram, the IN parameter behaves like a constant and cannot be assigned a value. An OUT parameter returns a value to the caller of a subprogram. Inside the subprogram, an OUT parameter behaves like a variable. An IN OUT parameter passes initial values to the subprogram which is being called, and returns updated values to the caller. Inside the subprogram, an IN OUT parameter acts like an initialized variable can be assigned a value and the value of the parameter can then be assigned to another variable. The actual parameter that corresponds to an IN OUT parameter must be a variable. It cannot be a constant or an expression (R, 2015).

2.13 The NOCOPY Hint

The NOCOPY hint tells the PL/SQL compiler to pass OUT and IN OUT parameters by reference, rather than by value. When parameters are passed by value, contents of OUT and IN OUT parameters are copied to temporary variables, which are then used by the subprogram which is being called. On completing the subprogram, values are copied back to the actual parameters but unhandled exceptions result in the original parameter values being left unchanged. The process of copying large parameters, such as records, collections, and objects, requires both time and memory which affects performance. With the NOCOPY hint, parameters are passed by reference and on successful completion the outcome is the same, but unhandled exceptions may leave parameters in an altered state (R, 2015).

2.14 The DETERMINISTIC Clause

The DETERMINISTIC keyword or clause can be used to indicate that the function will always return the same output or value for any given set of input argument values any point in time. The values returned by a deterministic function should not change even when the function is rewritten or recompiled. The performance benefit of deterministic functions is that if you call the function twice in a row with the same inputs, then Oracle has the option of 'remembering' the result from the first call to avoid actually executing the second call (R, 2015).

Here are examples of a simple function and a stored procedure in Oracle PL/SQL:

Example 1: Function

```
CREATE OR REPLACE FUNCTION calculate_area(  
  p_radius NUMBER  
) RETURN NUMBER IS  
  v_area NUMBER;  
BEGIN  
  -- Calculate the area of a circle  
  v_area := 3.14 * p_radius * p_radius;  
  RETURN v_area;  
END calculate_area;
```

In this example, a function named `calculate_area` is created. It takes a parameter `p_radius` (radius of a circle) and calculates the area of the circle using the formula πr^2 . The result is then returned.

You can call this function as follows:

```
DECLARE  
  radius NUMBER := 5;  
  area NUMBER;  
BEGIN  
  -- Call the function and store the result in the 'area' variable  
  area := calculate_area(radius);  
  
  -- Display the result  
  DBMS_OUTPUT.PUT_LINE('The area of the circle with radius ' || radius || ' is ' ||  
area);  
END;
```

Example 2: Stored Procedure

```
CREATE OR REPLACE PROCEDURE update_employee_salary(  
    p_employee_id NUMBER,  
    p_new_salary NUMBER  
) IS  
BEGIN  
    -- Update the salary of an employee  
    UPDATE employees  
    SET salary = p_new_salary  
    WHERE employee_id = p_employee_id;  
  
    -- Commit the transaction  
    COMMIT;  
END update_employee_salary;
```

In this example, a stored procedure named `update_employee_salary` is created. It takes two parameters, `p_employee_id` and `p_new_salary`, and updates the salary of the employee with the specified ID in the `employees` table.

You can call this stored procedure as follows:

```
DECLARE  
    employee_id NUMBER := 101;  
    new_salary NUMBER := 60000;  
BEGIN  
    -- Call the stored procedure to update the employee's salary  
    update_employee_salary(employee_id, new_salary);  
  
    DBMS_OUTPUT.PUT_LINE('Employee salary updated successfully.');
```

END;

2.15 Creating Triggers

Triggers are special PL/SQL program units that are stored in a database, and they are executed in response to certain events such as data modifications. They are necessary to automate tasks within the database. The modification statement may include INSERT, DELETE, and UPDATE statements, and there may be database-related statements such as ALTER, LOGON, and SHUTDOWN. Triggers can be disabled and enabled when necessary, depending on the bulk of modifications to be made, and can be triggered either before or after the execution of a statement. Thus, they allow you to perform various automated actions such as insertion, deletion, and alteration, further enhancing the database performance (R, 2015).

The Syntax of a Trigger

Here is the general syntax for a trigger:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n] [FOR EACH ROW]
WHEN (condition) BEGIN
END;
```

The following table lists and explains the different clauses in the header of a trigger.

Trigger Header Clause	Description
CREATE [OR REPLACE] TRIGGER trigger_name	Creates a trigger with the name specified or overwrites an existing trigger with the same name.
{BEFORE AFTER INSTEAD OF}	Specifies the trigger event.
{INSERT [OR] UPDATE [OR] DELETE}	Determines the triggering event.
[OF col_name]	Makes sure that the trigger fires only if the specified column is updated.
ON table_name	Associates the trigger with a table or view.
[REFERENCING OLD AS o NEW AS n]	Defines how the old and new values of the columns being changed must be referenced.
[FOR EACH ROW]	Specifies that the trigger must fire once for each row affected, instead of once for the entire statement.
WHEN (condition)	Filters the rows that must fire only for row-level triggers.

2.16 Trigger Types

Depending on the various tasks to be performed, triggers are categorized.

Table-level/Row-level: A table-level trigger is associated with a table and is executed when a specified action occurs on a table. These actions are performed by the INSERT, UPDATE, or DELETE statement. A single trigger may fire either before, or after these actions occur on the table.

The CREATE TRIGGER statement is used to create a table-level trigger. You must choose either BEFORE or AFTER, but not both to specify when to execute the trigger. BEFORE specifies that the trigger's action will execute before the DML statement that fired the trigger, and AFTER specifies that the trigger's action will execute after the triggering DML statement.

Event-level: An event-level trigger fires when certain events occur within the database, or the DDL statements are issued. It uses the ON [DATABASE | username.SCHEMA] clause to indicate whether the trigger will fire at the database or user level. It includes various triggers such as:

- The CREATE, ALTER, or DROP trigger that is fired when the specified DDL statements are issued.
- The SERVERERROR trigger that is fired when a major server error is encountered.
- The LOGON and LOGOFF triggers that are fired when users log in or log off the database.
- The STARTUP or SHUTDOWN trigger that is fired when STARTUP or SHUTDOWN commands are issued.

Compound Trigger: A compound trigger combines the features of row-level and event-level triggers (R, 2015).

2.17 Uses of Triggers

Triggers provide the standard capabilities to customize and manage data. Triggers are mainly used to generate values for columns based on sequences. They are used for auditing DML operations, preventing invalid transactions, and handling complex data verification operations. A trigger maintains synchronous table replicates; when you want to monitor transactions on your table it gathers statistics on table access (R, 2015).

2.17.1 Trigger States

A trigger can be in either of two states, enabled or disabled.

Enabled: A trigger that is enabled executes its trigger body if a triggering statement is entered and the trigger restriction (if any) evaluates to TRUE. This is the default state of a trigger.

Disabled: A trigger that is disabled does not execute its trigger body even if a triggering statement is entered and the trigger restriction (if any) evaluates to TRUE. The DISABLE clause of the CREATE TRIGGER statement is used to disable a trigger's state (R, 2015).

2.18 How to Create a Trigger

Here's an example of a simple trigger in Oracle. In this case, let's create a trigger that automatically updates a last_modified column whenever a row is updated in a table named your_table.

```
-- Assuming you have a table
CREATE TABLE your_table (
  id NUMBER PRIMARY KEY,
  data VARCHAR2(50),
  last_modified TIMESTAMP
);

-- Create or replace a trigger
CREATE OR REPLACE TRIGGER update_last_modified_trigger
BEFORE UPDATE ON your_table
FOR EACH ROW
BEGIN
  :NEW.last_modified := SYSTIMESTAMP;
END;
```

In this trigger:

update_last_modified_trigger is the trigger name.

BEFORE UPDATE indicates that the trigger should fire before an update operation.

FOR EACH ROW specifies that the trigger should be executed once for each row being updated.

Within the trigger body, :NEW.last_modified is used to reference the last_modified column of the row being updated, and SYSTIMESTAMP is used to set its value to the current timestamp.

Remember to replace your_table, id, data, and last_modified with your actual table and column names.

After creating this trigger, whenever you perform an update on the your_table table, the last_modified column will be automatically updated with the current timestamp.

```
UPDATE your_table SET data = 'Updated Data' WHERE id = 1;
```

Check the table to see the result:

```
SELECT * FROM your_table;
```

In Oracle, you can use the RAISE_APPLICATION_ERROR procedure within a trigger to raise a user-defined exception. Here's an example of a trigger that uses RAISE_APPLICATION_ERROR:

-- Assuming you have a table

```
CREATE TABLE your_table (  
  id NUMBER PRIMARY KEY,  
  data VARCHAR2(50),  
  check_value NUMBER  
);
```

-- Create or replace a trigger with RAISE_APPLICATION_ERROR

```
CREATE OR REPLACE TRIGGER check_value_trigger  
BEFORE INSERT OR UPDATE ON your_table  
FOR EACH ROW  
BEGIN  
  -- Check if check_value meets a certain condition  
  IF :NEW.check_value < 0 THEN  
    -- Raise an error if the condition is not met  
    RAISE_APPLICATION_ERROR(-20001, 'Check value must be non-negative');  
  END IF;  
END;
```


In this example:

check_value_trigger is the trigger name.

BEFORE INSERT OR UPDATE indicates that the trigger should fire before an insert or update operation.

FOR EACH ROW specifies that the trigger should be executed once for each row being inserted or updated.

Within the trigger body, it checks if the check_value column of the row being inserted or updated meets a certain condition. If the condition is not met (in this case, if check_value is less than 0), it raises an application error with the message 'Check value must be non-negative' and the error code -20001.

You can customize the condition and the error message based on your specific requirements.

Here's an example of how the trigger might be used:

-- This insert will raise an error

```
INSERT INTO your_table (id, data, check_value) VALUES (1, 'Test Data', -5);
```

The above insert will result in an error, and the error message will be displayed.

2.19 Create a Compound Trigger

You created a simple trigger to perform single data validations and manipulations in the data- base. But sometimes you need to execute multiple actions, which you can't do with a simple trigger. In this topic, you will create a compound trigger.

A simple trigger checks for a single condition. There may be instances where you may want to check for multiple conditions inside a single transaction. Here, you can create a compound trigger to check for multiple actions in your procedure (R, 2015).

2.20 Compound Triggers

Definition:

A compound trigger is a single trigger on a table enabling you to specify actions for each of four timing points: before a firing statement, before each row that the firing statement affects, after each row that the firing statement affects, and after the firing statement. A compound trigger has a section for each of its timing points. All of these sections can access a common PL/SQL state. The common state is established when the triggering statement starts and is destroyed when the triggering statement completes, regardless of the triggering statement causing an error (R, 2015).

```
CREATE OR REPLACE TRIGGER sal_check  
FOR update of sal ON emp_details COMPOUND TRIGGER  
new_sal varchar2(10);  
BEFORE STATEMENT IS /*The BEFORE statement-level trigger fires first*/ BEGIN  
new_sal:=0;
```

```

END BEFORE STATEMENT;
BEFORE EACH ROW IS /*The BEFORE row-level trigger fires next*/ BEGIN
if updating then
  dbms_output.put_line('datas are updated in the table for each row'); end if;
END BEFORE EACH ROW;
AFTER EACH ROW IS /*The AFTER row-level trigger fires once for each affected row*/
BEGIN
  new_sal:=new.sal;
END AFTER EACH ROW;
AFTER STATEMENT IS /*The AFTER statement-level trigger fires*/ BEGIN
if new_sal > 5000 then
  raise_application_error(-2000,'Total increase in salary cannot cross 5000');
end if;
END AFTER STATEMENT;
END;

```

/In the above example, if the salary of the employee id entered exceed 5000, an application error is raised.

2.21 Triggering Statements of Compound Triggers

The triggering statement of a compound trigger must be a DML statement. If the triggering statement does not affect any row, or if the compound trigger has neither a BEFORE STATEMENT section nor an AFTER STATEMENT section, then the trigger does not fire. The compound trigger is beneficial only when the triggering statement affects many rows (R, 2015).

A compound trigger is a type of trigger introduced in Oracle Database 11g to address certain challenges associated with triggers in earlier versions. In standard Oracle triggers, if you have multiple triggers for a specific timing point (like BEFORE INSERT, BEFORE UPDATE, etc.), each trigger is fired separately, potentially leading to performance and maintainability issues (R, 2015).

A compound trigger allows you to group multiple triggers into a single, more efficient structure. This helps in avoiding the global context switching overhead associated with multiple triggers and provides a way to share variables and state across multiple trigger actions (R, 2015).

A compound trigger consists of the following parts:

Declaration Section: This is where you declare variables that will be shared across all parts of the trigger.

Timing Point Sections (BEFORE EACH ROW, AFTER EACH ROW, etc.): These sections contain the actual trigger logic. You can have multiple sections for the same timing point, and all of them share the same variable scope.

Exception Handling Section: This section allows you to handle exceptions that may occur during trigger execution.

Here's a basic example of a compound trigger:

```
CREATE OR REPLACE TRIGGER compound_trigger_example  
FOR INSERT ON your_table  
COMPOUND TRIGGER  
  -- Declaration Section  
  v_counter NUMBER := 0;  
  
  BEFORE EACH ROW IS  
  BEGIN  
    -- Trigger logic BEFORE each row  
    v_counter := v_counter + 1;  
    DBMS_OUTPUT.PUT_LINE('Before Row ' || v_counter);  
  END BEFORE EACH ROW;  
  
  AFTER EACH ROW IS  
  BEGIN  
    -- Trigger logic AFTER each row  
    DBMS_OUTPUT.PUT_LINE('After Row ' || v_counter);  
  END AFTER EACH ROW;  
  
  AFTER STATEMENT IS  
  BEGIN  
    -- Trigger logic AFTER the entire statement  
    DBMS_OUTPUT.PUT_LINE('After Statement');  
  END AFTER STATEMENT;  
  
END
```

In this example, the compound trigger is defined for the BEFORE INSERT timing point. It has three sections: BEFORE EACH ROW, AFTER EACH ROW, and AFTER STATEMENT. The v_counter variable is declared in the declaration section and is shared across all sections.

This compound trigger prints messages before and after each row insertion and after the entire statement. Note that the BEFORE EACH ROW and AFTER EACH ROW sections share the same variable v_counter.

Compound triggers are particularly useful in scenarios where you need to maintain state or share information between different parts of the trigger. They can contribute to better performance and maintainability compared to having multiple separate triggers (R, 2015).

2.21.1 Triggering Statements of Compound Triggers

The triggering statement of a compound trigger must be a DML statement. If the triggering statement does not affect any row, or if the compound trigger has neither a BEFORE STATEMENT section nor an AFTER STATEMENT section, then the trigger does not fire. The compound trigger is beneficial only when the triggering statement affects many rows (R, 2015).

2.21.2 Compound Trigger Sections

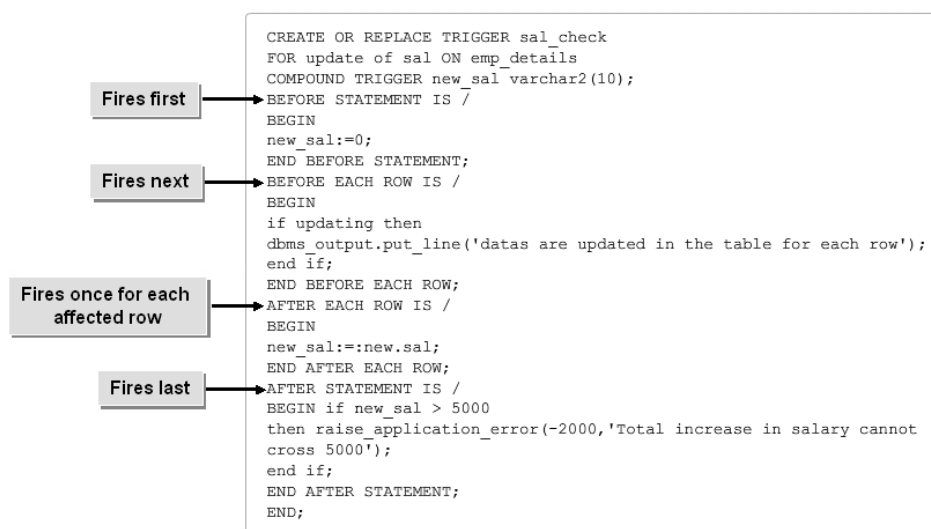
A compound trigger section that is defined on a table can have one or more timing points.

Section	Timing Point
BEFORE STATEMENT	Before the triggering statement executes
AFTER STATEMENT	After the triggering statement executes.
BEFORE EACH ROW	Before each row that the triggering statement affects
AFTER EACH ROW	After each row that the triggering statement affects.

2.21.3 Trigger Execution

When you have multiple triggers against a table for the same triggering event (INSERT, UPDATE, or DELETE), Oracle follows a specific execution hierarchy. The BEFORE statement- level trigger fires first. The BEFORE row-level trigger fires next, once for each row affected (R, 2015).

The AFTER row-level trigger fires once for each affected row and alternates with the BEFORE row-level trigger. Finally, the AFTER statement-level trigger fires (R, 2015).



3 Practical Exercises

Run the preload given to you by the lecturer for the 2017 Exam for Semester 1 to create all the tables and do the questions below:

- Create a trigger called `Offence_Entry` that will not allow an offence rating to be entered into the offence table that is less than zero or greater than 10. In your query include the code to test the trigger.
- Create a procedure called `Criminal_Details` that will accept a criminal ID as an input parameter and will display the criminal name and the offence committed on a certain day. In your solution display the data for any criminal and provide the code to execute the procedure with an exception handling if no criminal data is found.

Sample Results:

CRIMINAL DETAILS: Sam Jackson committed a House Robbery on the 15/OCT/17.

- Create a function called `Case_Adjustments` that will accept a criminal ID as an input parameter and add five (5) days to the criminal's case date. In your query use any criminal ID as the input parameter and display the criminal name, offence committed and the new case date.

Sample Results:

Sam, Jackson committed the offence House Robbery and has a new case date which is 20/OCT/17.

Learning Unit 9: Advanced Interface Methods and Security

Learning objectives addressed in this learning unit:

- LO1: Describe external procedures.
- LO2: Discuss the benefits of external procedures.
- LO3: Explain how to execute external C programs from PL/SQL.
- LO4: Execute Java programs from PL/SQL.
- LO5: Create and Interface the database to an organisational platform.
- LO6: Create and use packages;
- LO7: Describe the importance of database security.
- LO8: Discuss the common threats and challenges to databases.
- LO9: Explore the best practices in database security.
- LO10: Describe various data protection tools and platforms.

My notes

1 Introduction

You have learned to create applications that use collections. To execute the PL/SQL subprogram that invokes the Java class method, we need to know about the advanced interface methods that include calling C and Java from PL/SQL function. In this lesson, you will implement an external C routine from PL/SQL code and how to incorporate Java code into your PL/SQL programs (R, 2015).

Application designing philosophy demands the effortful integration of client platform and data- base to build up a concrete standard. Oracle has emerged out as a successful database for C, C++, JAVA, Visual basic, COBOL, and other contemporary generation languages. Learning to derive the benefits of other programming languages such as C and Java will further strengthen up your PL/SQL code (R, 2015).

2 External Procedures Overview

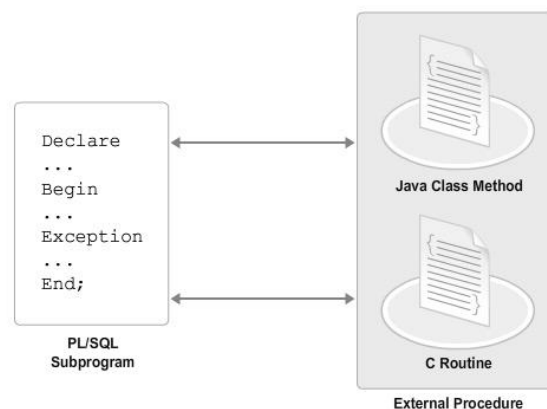
There might be many instances where you may require an external program to boost the logical design of a program. In this topic, you will familiarize yourself with external procedures.

Imagine that your organization uses complicated statistics programs written in a language other than PL/SQL. The customer wants to access the data stored in an Oracle database and pass the data into this particular program. To do this, you need to externally compile other programs from within PL/SQL. After execution of a program data is inserted into the appropriate oracle database using PL/SQL depending on the result of the evaluations (R, 2015).

2.1 External Procedures

An external procedure is a program written in a non PL/SQL programming language. To avoid any confusion with “PL/SQL procedures,” the programs from non PL/SQL background are termed as external procedures. External procedures are also known as an external routine (R, 2015).

External programs are more frequently used in applications of computation purpose, where a JAVA or C++ based logic can boost up the logical design. In addition, they not only add up server side programming feature, but also a routine as an external procedure acts as interface between the server and external sources (R, 2015).



The Need for an External Procedure

An external procedure enables you to:

- Move computation-bound programs from the client to the server since it enables faster execution.
- Interface the database server with external systems and data sources.
- Extend the functionality of the database itself (R, 2015).

2.2 External Procedures in PL/SQL

An external procedure is stored in a dynamic link library (DLL), shared object (.so file in UNIX), or **libunit** in the case of a Java class method that can perform special purpose processing. You can first publish the routine in the base language, and then call it to perform processing. You can then call the external routine from within PL/SQL or SQL. With C, you publish the routine through a library schema object, which is called from PL/SQL, that contains the compiled library file name that is stored on the operating system. With Java, publishing the routine is accomplished through creating a class **libunit**. There are two relevant terms “call out” and “call back” associated with the external routine handling. A call is referred as callout when PL/SQL procedure invokes an external procedure. On the other hand, if the external procedure invokes a statement, which drives the database engine, the call is known as callback. The statement can be a SQL or PL/SQL construct, which hits the oracle server to perform an operation (R, 2015).

Oracle server supports the execution of external procedures as shared library unit. Once the external routine or procedure is developed, the compiler generates the corresponding library format, which contains the external routine. Depending on the base language and compiler standards, the shared unit is known by a different name; conceptually it is the same as DLL (Dynamic Link Library). Thereafter the routine has to be published through a wrapper PL/SQL program (R, 2015).

2.3 Benefits of External Procedures

There are several benefits to using external procedures:

- Integration of strengths: The realization of capabilities of a programming language in another one demonstrates flexibility of one and adaptation of the other. Also a program adding up to the features of another language integrates the strengths and capabilities of programming.
- Reusability of client logic: Since the server side external program is sharable among all the database users, the logic could be reused by the connecting user.
- Logical extensibility: External routines maintain the margin to extend its logic. From an application's perspective, external procedures also avoid logical redundancy.
- Enhanced Performance: Moving the execution of calculative programs and methods from client to server side improve their execution by reducing the network round trips (R, 2015).

2.4 Execute External C Programs from PL/SQL

In the previous topic, you familiarized yourself with external procedures. In order to blend your existing knowledge in C programming with PL/SQL, you need to execute external C programs.

C is one of the most widely used programming languages of all time and there are very few computer architectures for which a C compiler does not exist. There may be times when a program written in C cannot be replicated in PL/SQL. In such instances, learning to execute C programs from within PL/SQL assumes a high level of importance (R, 2015).

2.5 External C Procedures

Different units of code written in C is known as external C procedures. Calling an external program inside Oracle needs a program which must run as a shared library. Shared library refers to an operating system file that stores the external procedure. This type of program is called as a DLL file (Dynamically Linked Library) on Microsoft operating systems like Solaris, AIX, and LINUX. The process of executing external procedures is called extproc process (R, 2015).

2.6 External C Procedure Components

External C procedures consist of a number of components.

Component	Description
External procedure	A unit of code written in C
Shared library	An operating system file that stores the external procedure
Alias library	A schema object that represents the operating system shared library
PL/SQL subprograms	Packages, procedures, or functions that define the program unit specification and mapping to the PL/SQL library
extproc process	A session-specific process that executes external procedures
Listener process	A process that starts the extproc process and assigns it to the process executing the PL/SQL subprogram

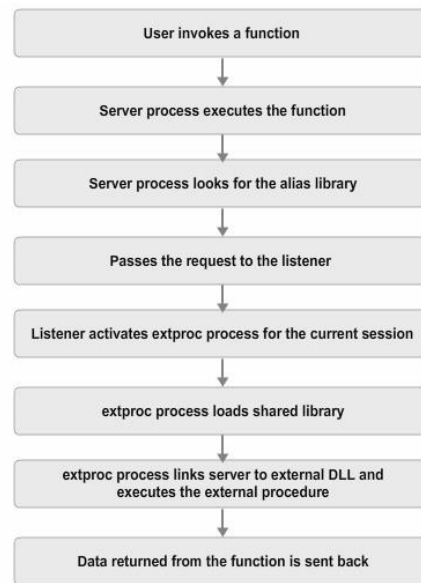
2.7 The extproc Process

The extproc process services the execution of external procedures for the duration of the session until the user logs off. Each session uses a different extproc process to execute external procedures. The listener must be configured to allow the server to be associated with the extproc process. The listener must be on the same machine as the server (R, 2015).

2.8 External Procedure Execution Process

A process is carried out by the database engine when a call specification containing the external procedure is invoked.

1. The user invokes a function.
2. Server processes execute the function.
3. Server process looks for the alias directory.
4. The server passes the request to the listener.
5. The listener activates extproc process for the current session.
6. The extproc process loads the shared library.
7. The extproc process links the server to the external DLL and executes the external procedure.
8. Data returned from the function is sent back.



2.9 Execute Java Programs from PL/SQL

You have now learned to execute external C programs from PL/SQL. Now you may need to do your coding program in Java. In this topic, you will execute Java programs from PL/SQL.

Java is currently one of the most popular programming languages in use, particularly for client-server web applications. For instance, if you need to write a program in a method that accepts a 16-digit credit card number as the argument and returns the formatted credit card number (4 digits followed by a space), you will definitely need Java. Therefore, learning to execute external Java programs from within PL/SQL is extremely important (R, 2015).

2.10 Java Virtual Machine

Definition:

A Java virtual machine (JVM) provides an environment which interprets a compiled Java binary code and executes it. A JVM executes the Java code as an operating system which makes it independent of an operating system. This feature of JVM makes it capable of handling OS requests which are required for the execution of a Java program like Socket connection from a client machine to a remote machine (R, 2015).

Example:

A request to establish a socket connection to a remote machine will involve an operating system call. Different operating systems can handle sockets in different ways. The programmer need not worry about such details. It is the responsibility of the JVM to handle these translations so that the operating system and CPU architecture are completely irrelevant to the developer (R, 2015).

2.11 Oracle JVM

The Oracle JVM is a combination of standard Java language and Java Virtual Machine specification. The Oracle JVM consists of CORBA-compliant Object Request Broker (ORB), and the Oracle JVM. They both support the Enterprise JavaBeans standard. The Oracle JVM helps increase scalability and performance of Java stored programs that runs in the same process and memory space as the database kernel. As an add-on advantage, it provides access to the standard Java class libraries such as java.lang, java.util, and java.math (R, 2015).

2.12 JDBC Drivers

Definition:

A Java application that communicates with a database is called a JDBC driver. To connect with individual databases, JDBC (the Java Database Connectivity API) needs drivers for each database. The JDBC driver enables the protocol for the transfer of query and result between the client and database. JDBC translates its query into a corresponding ODBC query, which is then handled by the ODBC driver (R, 2015).

Example:

```
import java.sql.*; import java.util.*; class InsertJDBCDemo{
public static void main(String arg[]){ String emp_Name;
int emp_Id; int val=0;
Connection con=null; PreparedStatement st;
Scanner sc=new Scanner(System.in); System.out.println("Enter the Employee
Name"); emp_Name=sc.next();
System.out.println("Enter the Employee Id"); emp_Id=sc.nextInt();
try
{

// connect to the database
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); con=
DriverManager.getConnection("jdbc:odbc:mydsn","hr","people")
;
}

catch(ClassNotFoundException cnf)
{System.err.println(cnf.getMessage());}
catch(SQLException se){System.err.println(se.getMessage());}
try {

// Inserting the data into the database st=con.prepareStatement("Insert into Emp
values (?,?)
");
}
st.setString(1,emp_Name); st.setInt(2,emp_Id); val=st.executeUpdate();
System.out.println(val+" Record Inserted");
```

```

catch(SQLException sqe){System.err.println(sqe.getMessage())
;}
    }
}

```

2.13 SQLJ

Definition:

SQLJ is basically a JAVA program that consists of embedded SQL statements interacting with the ISO. SQLJ code is a mixture of standard Java source, class declarations, and executable statements with embedded SQL operations. SQLJ uses simplified codes for database access. It handles all the static SQL operations and provides extensions to support dynamic SQL operations. Its application is more of static SQL than dynamic SQL. SQLJ has both translator and a run time compiler which is easily adapted into the operating developing environment. The embedded SQL is replaced by the translation process which is then implemented into the SQL operations. SQLJ is performed through calls to a JDBC driver (R, 2015).

Example:

```

import sqlj.runtime.*; import sqlj.runtime.ref.*; import java.sql.*;
import java.io.*;
import oracle.sqlj.runtime.*; public class DisplaySQLJDemo {
public static void main(String [] args) { String Name;
int Id;

try {
// connect to the database Oracle. connect("jdbc:
oracle:thin:@localhost:1521:orcl","hr","people" );
// get the employee name for empid in the database emp #sql { SELECT emp_Name
INTO :name FROM Emp WHERE
emp_Id= :id };

// display message
System. out.println("Employee ID:"+id); System. out.println("Employee
Name:"+name);

}
catch ( SQLException e ) {          System. err.println("SQLException " + e);      }
finally {
try {
// disconnect from the database Oracle. close( );
} catch ( SQLException e ) {
System. err.println("SQLException " + e);
}
}
}
}

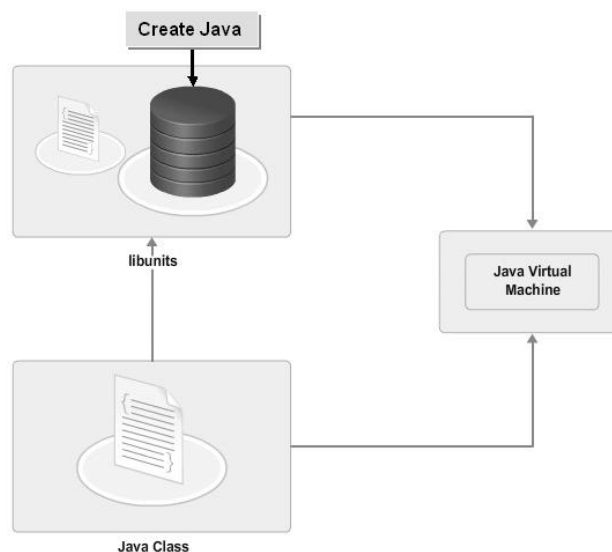
```

}

2.14 Calling a Java Class Method from PL/SQL

When a Java class is loaded into database from command link utility, JVM acknowledges the request. It loads the java binaries and resources from OS location to the database library units. This operation is similar to the generation of shared library units in External C program access through PL/SQL. The various steps involved in calling a Java class method from PL/SQL are:

1. Load the Java source or Class file from command line editor to the database with the help of “loadjava” utility. Once they are loaded into the database, they are stored as Java stored objects in the Schema.
2. Create a Call specification or a PL/SQL subprogram to publish the Java program. A Call specification is required to map with the Java method. A Call specification can be Oracle stored Procedure or a function.
3. Test the PL/SQL block to verify the execution of Java Class method. Since the Call specification has been published, the database engine is ready to communicate with JVM and execute Java class from PL/SQL (R, 2015).



2.15 The Importance of Database Security

Database security is of paramount importance in the realm of information technology due to several critical reasons:

Confidentiality

Sensitive Data Protection: Databases often store sensitive information such as personal details, financial records, and proprietary business data. Ensuring the confidentiality of this information is crucial to prevent unauthorized access and data breaches (Gaetjen, Knox & Maroulis, 2015).

Integrity

Data Accuracy and Reliability

Database security measures ensure that data remains accurate and reliable. Unauthorized modifications, deletions, or insertions can compromise the integrity of the data, leading to misinformation and potential business losses (Gaetjen *et al*, 2015).

Availability

Preventing Denial of Service (DoS) Attacks

Database security helps in safeguarding against denial of service attacks that aim to disrupt services or make data unavailable. Ensuring the availability of the database is essential for business continuity.

Regulatory Compliance:

Meeting Legal Requirements: Many industries and jurisdictions have regulations that mandate the protection of sensitive information (Gaetjen *et al*, 2015).

Preventing Unauthorized Access

User Authentication and Authorization: Robust database security involves implementing strong authentication mechanisms to verify the identity of users. Authorization controls ensure that users have appropriate access rights, limiting access to only the necessary data and functionalities (Gaetjen *et al*, 2015).

Data Encryption

Securing Data in Transit and at Rest

Encryption helps protect data both during transmission over networks and when stored on disks. This prevents eavesdropping on sensitive information and adds an extra layer of security in case of unauthorized physical access to storage media.

Auditing and Monitoring

Detection of Anomalies

Database security includes features for auditing and monitoring user activities. This helps detect and respond to unusual or suspicious behaviour, allowing organizations to take proactive measures against potential security threats.

Preventing SQL Injection and Other Attacks

Secure Coding Practices: Database security involves promoting secure coding practices to prevent common vulnerabilities like SQL injection. By validating and sanitizing input, organizations can reduce the risk of malicious exploitation of vulnerabilities (Gaetjen et al, 2015).

Data Backup and Recovery

Disaster Recovery Planning: Database security encompasses strategies for regular data backups and recovery plans. In the event of data loss or corruption, having reliable backup mechanisms ensures the ability to restore critical information and maintain business operations.

In summary, database security is essential for safeguarding sensitive information, maintaining the integrity and availability of data, complying with regulations, and building trust with stakeholders. A comprehensive approach to database security involves a combination of technological solutions, policies, and user education to create a robust defence against evolving cybersecurity threats (Gaetjen et al, 2015).

2.16 Common threats and challenges to databases

Databases face a range of threats and challenges, and addressing these is crucial to maintaining the confidentiality, integrity, and availability of data. Here are some common threats and challenges to databases:

Unauthorized Access

Threat: Unauthorized individuals gaining access to the database can lead to data breaches, unauthorized modifications, or theft of sensitive information.

Challenge: Implementing strong authentication mechanisms, access controls, and encryption helps mitigate the risk of unauthorized access (Gaetjen et al, 2015).

SQL Injection

Threat: SQL injection attacks involve injecting malicious SQL code into input fields, potentially leading to unauthorized access or manipulation of the database.

Challenge: Validating and sanitizing user inputs, using parameterized queries, and employing web application firewalls can help prevent SQL injection attacks.

Data Breaches

Threat: Data breaches can result from various factors, including hacking, insider threats, or inadvertent exposure of sensitive information.

Challenge: Implementing robust security measures, encryption, and regularly monitoring and auditing database activities can help detect and respond to potential breaches.

Malware and Ransomware

Threat: Malicious software can infect databases, leading to data corruption, theft, or ransomware attacks that encrypt data and demand payment for decryption.

Challenge: Regularly updating antivirus software, implementing intrusion detection systems, and maintaining up-to-date backups are crucial for combating malware threats.

Insider Threats

Threat: Employees or other trusted individuals with access to the database may intentionally or unintentionally misuse their privileges.

Challenge: Implementing least privilege principles, conducting regular access reviews, and monitoring user activities can help mitigate insider threats.

Lack of Encryption

Threat: Data transmitted over networks or stored on disks without encryption is vulnerable to interception or unauthorized access.

Challenge: Implementing encryption protocols for data in transit (TLS/SSL) and data at rest (disk encryption) helps protect against eavesdropping and unauthorized access.

Data Redundancy and Inconsistency

Threat: Redundant or inconsistent data can result from errors in database design, leading to inaccuracies and difficulties in maintaining data integrity.

Challenge: Ensuring proper normalization, enforcing data validation rules, and implementing transaction management practices help maintain data consistency.

Inadequate Backup and Recovery

Threat: Failure to regularly back up data and establish effective recovery plans can result in data loss in the event of hardware failures, disasters, or other unforeseen incidents.

Challenge: Regularly performing backups, testing recovery procedures, and maintaining off-site backups contribute to effective data recovery capabilities.

Data Interception

Threat: Data transmitted over insecure networks is susceptible to interception by attackers, leading to potential exposure of sensitive information.

Challenge: Implementing secure communication protocols (e.g., HTTPS), using virtual private networks (VPNs), and encrypting sensitive data during transmission help mitigate the risk of data interception.

Inadequate Patch Management

Threat: Failure to apply security patches and updates can leave databases vulnerable to known vulnerabilities that attackers may exploit.

Challenge: Establishing a proactive patch management process, staying informed about security updates, and regularly applying patches help reduce the risk of exploitation.

Addressing these threats and challenges requires a comprehensive approach, involving a combination of technological solutions, security best practices, regular monitoring, and user education. Regularly assessing and updating security measures is essential to stay ahead of evolving threats in the dynamic landscape of database security.

2.17 Best practices in Database Security

Implementing effective database security is crucial to safeguarding sensitive information and ensuring the integrity, confidentiality, and availability of data. Here are some best practices in database security:

Authentication and Authorization

Use Strong Passwords: Enforce strong password policies for database accounts, including a mix of uppercase and lowercase letters, numbers, and special characters.

Implement Multi-Factor Authentication (MFA): Require additional authentication factors, such as tokens or biometrics, to enhance user verification.

Least Privilege Principle: Grant users the minimum level of access necessary to perform their tasks. Regularly review and update user privileges (Gaetjen *et al*, 2015).

Data Encryption

Encrypt Data in Transit: Use secure communication protocols (e.g., TLS/SSL) to encrypt data transmitted between the application and the database server.

Encrypt Data at Rest: Implement encryption mechanisms to protect data stored on disks or in backups, preventing unauthorized access in case of physical theft (Gaetjen *et al*, 2015).

Regularly Update and Patch

Apply Security Patches: Keep the database management system (DBMS) and associated software up to date by promptly applying security patches and updates.

Regularly Update Third-Party Components: Ensure that any third-party tools or components integrated with the database are also kept up to date (Gaetjen *et al*, 2015).

Auditing and Monitoring

Audit Database Activities: Enable auditing features to log and monitor database activities, including login attempts, privilege changes, and data access.

Implement Real-time Monitoring: Employ tools to monitor database activity in real time, enabling the detection of anomalies or suspicious behaviour (Gaetjen *et al*, 2015).

Secure Coding Practices

Prevent SQL Injection: Use parameterized queries and input validation to prevent SQL injection attacks. Educate developers on secure coding practices.

Input Validation: Validate and sanitize user inputs to prevent common vulnerabilities such as injection attacks and cross-site scripting (XSS).

Backup and Recovery

Regularly Back Up Data: Establish regular backup schedules for database contents. Ensure that backups are stored securely and are regularly tested for recoverability.

Disaster Recovery Planning: Develop and maintain a comprehensive disaster recovery plan to minimize downtime and data loss in the event of a disaster (Gaetjen *et al*, 2015).

Database Firewall

Implement a Database Firewall: Use a database firewall to monitor and control database access, detect and prevent unauthorized activities, and provide an additional layer of defence against attacks.

User Education

Security Awareness Training: Provide regular security awareness training to database users and administrators. Educate them about security risks, best practices, and how to recognize and report security incidents.

Data Masking and Redaction

Mask Sensitive Data: Implement data masking to conceal sensitive information when displayed to users who do not require full access.

Redact Confidential Information: Redact or anonymize personally identifiable information (PII) and other confidential data in reports or logs (Gaetjen *et al*, 2015).

Regular Security Assessments

Conduct Regular Security Audits: Perform regular security assessments and audits of the database environment to identify vulnerabilities and weaknesses.

Penetration Testing: Engage in penetration testing to simulate real-world attacks and assess the effectiveness of security measures (Gaetjen *et al*, 2015).

Secure Configuration

Follow Security Best Practices: Configure the database server and associated components according to security best practices provided by the database vendor.

Remove Unused Features: Disable or remove unnecessary database features and functionalities to reduce the attack surface (Gaetjen *et al*, 2015).

Incident Response Plan

Develop an Incident Response Plan: Have a well-defined incident response plan in place to efficiently respond to and mitigate security incidents. This includes communication protocols and steps for containment and recovery.

By incorporating these best practices into your database security strategy, you can establish a robust defence against a variety of threats and vulnerabilities. Remember that database security is an ongoing process that requires regular assessments, updates, and proactive measures to adapt to evolving security landscapes.

2.18 Data protection tools and platforms

Data protection tools and platforms play a crucial role in securing sensitive information, preventing data breaches, and ensuring compliance with privacy regulations. Here are descriptions of various types of data protection tools and platforms:

Encryption Tools

Purpose: Encrypting data helps protect it from unauthorized access by converting it into a format that can only be read with a decryption key.

Examples:

Symmetric Encryption Tools: AES Crypt, 3DES

Asymmetric Encryption Tools: PGP (Pretty Good Privacy), GPG (GNU Privacy Guard)

Database Encryption Tools: Oracle Transparent Data Encryption (TDE), Microsoft SQL Server Always Encrypted (Gaetjen *et al*, 2015).

Data Loss Prevention (DLP) Platforms

Purpose: DLP platforms monitor, detect, and prevent the unauthorized transmission or exfiltration of sensitive data, both within and outside the organization.

Examples: Symantec Data Loss Prevention, McAfee Total Protection for DLP, Digital Guardian (Gaetjen *et al*, 2015).

Tokenization Solutions:

Purpose: Tokenization replaces sensitive data with tokens, preserving the format but eliminating the actual content, reducing the risk of data exposure.

Examples: Voltage SecureData, TokenEx, Thales CipherTrust Tokenization

Database Activity Monitoring (DAM) Tools:

Purpose: DAM tools monitor database activity in real-time, providing insights into user actions, detecting suspicious behavior, and helping to enforce security policies.

Examples: IBM Guardium, Imperva SecureSphere Database Security, McAfee Database Security (Gaetjen *et al*, 2015).

Secure File Transfer Tools

Purpose: These tools enable secure and encrypted file transfers, ensuring that sensitive data remains protected during transit.

Examples: Axway SecureTransport, Globalscape EFT, IBM Aspera (Gaetjen *et al*, 2015).

Endpoint Security Platforms

Purpose: Endpoint security platforms protect individual devices (computers, laptops, mobile devices) from various security threats, including data breaches.

Examples: Symantec Endpoint Protection, McAfee Endpoint Security, CrowdStrike Falcon (Gaetjen *et al*, 2015).

Identity and Access Management (IAM) Solutions

Purpose: IAM solutions manage and secure user identities and access rights, ensuring that only authorized users have appropriate access to data.

Examples: Okta, Microsoft Azure Active Directory, ForgeRock (Gaetjen *et al*, 2015).

Backup and Disaster Recovery Platforms

Purpose: These platforms facilitate regular data backups and provide mechanisms for disaster recovery to minimize downtime and data loss.

Examples: Veeam Backup & Replication, Commvault, Rubrik (Gaetjen *et al*, 2015).

Data Masking Tools

Purpose: Data masking tools obscure or anonymize sensitive information, allowing organizations to use realistic data for testing and development without exposing confidential details.

Examples: Delphix, Informatica Persistent Data Masking, IBM Guardium Data Masking (Gaetjen *et al*, 2015).

Security Information and Event Management (SIEM) Platforms

Purpose: SIEM platforms collect and analyze security event data from various sources to identify and respond to security incidents.

Examples: Splunk, IBM QRadar, ArcSight (Gaetjen *et al*, 2015).

Cloud Security Platforms

Purpose: With the rise of cloud computing, these platforms provide security services designed to protect data and applications hosted in cloud environments.

Examples: AWS Security Hub, Azure Security Center, Google Cloud Security Command Center (Gaetjen *et al*, 2015).

Mobile Device Management (MDM) Solutions

Purpose: MDM solutions manage and secure mobile devices, ensuring that sensitive data on smartphones and tablets is protected.

Examples: MobileIron, VMware Workspace ONE, Microsoft Intune (Gaetjen *et al*, 2015).

Web Application Firewalls (WAF)

Purpose: WAFs protect web applications from various online threats, including SQL injection, cross-site scripting (XSS), and other security vulnerabilities.

Examples: Imperva Web Application Firewall, F5 Networks BIG-IP Application Security Manager, Cloudflare WAF (Gaetjen *et al*, 2015).

Endpoint Encryption Tools

Purpose: These tools focus on encrypting data on individual endpoints (devices) to protect information stored locally.

Examples: BitLocker (Microsoft), FileVault (Apple), Symantec Endpoint Encryption (Gaetjen *et al*, 2015).

Privacy Management Platforms

Purpose: Privacy management platforms help organizations comply with data protection regulations by managing consent, handling subject access requests, and ensuring privacy by design.

Examples: OneTrust, TrustArc, WireWheel (Gaetjen *et al*, 2015).

Choosing the right combination of these tools and platforms depends on the specific needs and security requirements of an organization. Many organizations adopt a multi-layered approach, combining several tools to create a robust and comprehensive data protection strategy.

3 Revision Exercises

- Describe the importance of database security;
- Discuss the common threats and challenges to databases;
- Explore the best practices in database security;
- Describe various data protection tools and platforms.

4 Recommended Additional Reading

Gaetjen, S., Knox, D. and Maroulis, W. (2015) Oracle Database 12C security, O'Reilly Online Learning. Available at: <https://www.oreilly.com/library/view/oracle-database-12c/9780071824286/> (Accessed: 27 November 2023).

Oracle Applications User's Guide. Available at: https://docs.oracle.com/cd/A60725_05/pdf/oaug.pdf (Accessed: 27 November 2023).

PL/SQL tutorial (no date) Online Tutorials, Courses, and eBooks Library. Available at: <https://www.tutorialspoint.com/plsql/index.htm> (Accessed: 27 November 2023).

R, Anitha. 2015. Oracle Database 11g [with annexures] [Module Manual]: IIE80-063sq_rev3.0. Johannesburg: Masterskill.

Bibliography

Gaetjen, S., Knox, D. and Maroulis, W. (2015) Oracle Database 12C security, O'Reilly Online Learning. Available at: <https://www.oreilly.com/library/view/oracle-database-12c/9780071824286/> (Accessed: 27 November 2023).

Oracle Applications User's Guide. Available at: https://docs.oracle.com/cd/A60725_05/pdf/oaug.pdf (Accessed: 27 November 2023).

PL/SQL tutorial (no date) Online Tutorials, Courses, and eBooks Library. Available at: <https://www.tutorialspoint.com/plsql/index.htm> (Accessed: 27 November 2023).

R, Anitha. 2015. Oracle Database 11g [with annexures] [Module Manual]: IIE80-063sq_rev3.0. Johannesburg: Masterskill.

Intellectual Property

Plagiarism occurs in a variety of forms. Ultimately though, it refers to the use of the words, ideas or images of another person without acknowledging the source using the required conventions. The IIE publishes a Quick Reference Guide that provides more detailed guidance, but a brief description of plagiarism and referencing is included below for your reference. It is vital that you are familiar with this information and the Intellectual Integrity Policy before attempting any assignments.

Introduction to Referencing and Plagiarism

What is 'Plagiarism'?

'Plagiarism' is the act of taking someone's words or ideas and presenting them as your own.

What is 'Referencing'?

'Referencing' is the act of citing or giving credit to the authors of any work that you have referred to or consulted. A 'reference' then refers to a citation (a credit) or the actual information from a publication that is referred to.

Referencing is the acknowledgment of any work that is not your own, but is used by you in an academic document. It is simply a way of giving credit to and acknowledging the ideas and words of others.

When writing assignments, students are required to acknowledge the work, words or ideas of others through the technique of referencing. Referencing occurs in the text at the place where the work of others is being cited, and at the end of the document, in the bibliography.

The bibliography is a list of all the work (published and unpublished) that a writer has read in the course of preparing a piece of writing. This includes items that are not directly cited in the work.

A reference is required when you:

- Quote directly: when you use the exact words as they appear in the source;
- Copy directly: when you copy data, figures, tables, images, music, videos or frameworks;
- Summarise: when you write a short account of what is in the source;
- Paraphrase: when you state the work, words and ideas of someone else in your own words.

It is standard practice in the academic world to recognise and respect the ownership of ideas, known as intellectual property, through good referencing techniques. However, there are other reasons why referencing is useful.

Good Reasons for Referencing

It is good academic practice to reference because:

- It enhances the quality of your writing;
- It demonstrates the scope, depth and breadth of your research;
- It gives structure and strength to the aims of your article or paper;
- It endorses your arguments;
- It allows readers to access source documents relating to your work, quickly and easily.

Sources

The following would count as 'sources':

- Books,
- Chapters from books,
- Encyclopaedias,
- Articles,
- Journals,
- Magazines,
- Periodicals,
- Newspaper articles,
- Items from the Internet (images, videos, etc.),
- Pictures,
- Unpublished notes, articles, papers, books, manuscripts, dissertations, theses, etc.,
- Diagrams,
- Videos,
- Films,
- Music,
- Works of fiction (novels, short stories or poetry).

What You Need to Document from the Hard Copy Source You are Using

(Not every detail will be applicable in every case. However, the following lists provide a guide to what information is needed.)

You need to acknowledge:

- The words or work of the author(s),
- The author(s)'s or editor(s)'s full names,
- If your source is a group/ organisation/ body, you need all the details,
- Name of the journal, periodical, magazine, book, etc.,
- Edition,
- Publisher's name,
- Place of publication (i.e. the city of publication),
- Year of publication,
- Volume number,
- Issue number,
- Page numbers.

What You Need to Document if you are Citing Electronic Sources

- Author(s)'s/ editor(s)'s name,
- Title of the page,
- Title of the site,
- Copyright date, or the date that the page was last updated,
- Full Internet address of page(s),
- Date you accessed/ viewed the source,
- Any other relevant information pertaining to the web page or website.

Referencing Systems

There are a number of referencing systems in use and each has its own consistent rules. While these may differ from system-to-system, the referencing system followed needs to be used consistently, throughout the text. Different referencing systems cannot be mixed in the same piece of work!

A detailed guide to referencing, entitled Referencing and Plagiarism Guide is available from your library. Please refer to it if you require further assistance.

When is Referencing Not Necessary?

This is a difficult question to answer – usually when something is 'common knowledge'. However, it is not always clear what 'common knowledge' is.

Examples of 'common knowledge' are:

- Nelson Mandela was released from prison in 1990;
- The world's largest diamond was found in South Africa;
- South Africa is divided into nine (9) provinces;
- The lion is also known as 'The King of the Jungle'.
- $E = mc^2$
- The sky is blue.

Usually, all of the above examples would not be referenced. The equation $E = mc^2$ is Einstein's famous equation for calculations of total energy and has become so familiar that it is not referenced to Einstein.

Sometimes what we think is 'common knowledge', is not. For example, the above statement about the sky being blue is only partly true. The light from the sun looks white, but it is actually made up of all the colours of the rainbow. Sunlight reaches the Earth's atmosphere and is scattered in all directions by all the gases and particles in the air. The smallest particles are by coincidence the same length as the wavelength of blue light. Blue is scattered more than the other colours because it travels as shorter, smaller waves. It is not entirely accurate then to claim that the sky is blue. It is thus generally safer to always check your facts and try to find a reputable source for your claim.

Important Plagiarism Reminders

The IIE respects the intellectual property of other people and requires its students to be familiar with the necessary referencing conventions. Please ensure that you seek assistance in this regard before submitting work if you are uncertain.

If you fail to acknowledge the work or ideas of others or do so inadequately this will be handled in terms of the Intellectual Integrity Policy (available in the library) and/ or the Student Code of Conduct – depending on whether or not plagiarism and/ or cheating (passing off the work of other people as your own by copying the work of other students or copying off the Internet or from another source) is suspected.

Your campus offers individual and group training on referencing conventions – please speak to your librarian or ADC/ Campus Co-Navigator in this regard.

Reiteration of the Declaration you have signed:

1. I have been informed about the seriousness of acts of plagiarism.
2. I understand what plagiarism is.
3. I am aware that The Independent Institute of Education (IIE) has a policy regarding plagiarism and that it does not accept acts of plagiarism.
4. I am aware that the Intellectual Integrity Policy and the Student Code of Conduct prescribe the consequences of plagiarism.

5. I am aware that referencing guides are available in my student handbook or equivalent and in the library and that following them is a requirement for successful completion of my programme.
6. I am aware that should I require support or assistance in using referencing guides to avoid plagiarism I may speak to the lecturers, the librarian or the campus ADC/ Campus Co-Navigator.
7. I am aware of the consequences of plagiarism.

Please ask for assistance prior to submitting work if you are at all unsure.